Amiga® Mail

Volume II

The Commodore® Amiga Technical Newsletter

July/August 1992

Copyright © 1992 by Commodore-Amiga, Inc, 1200 Wilson Drive, West Chester, PA, 19380. All rights reserved. Printed in the USA. Authors' materials used herein have been reproduced with the permission of the authors indicated. Use or reproduction of such materials shall be in accordance with the authors' instructions. Commodore and CDTV are registered trademarks of Commodore Electronics Ltd. Amiga is a registered trademark of Commodore-Amiga, Inc. Kickstart, Workbench, AmigaDOS, Amiga 500, Amiga 2000, and Amiga 3000 are trademarks of Commodore-Amiga, Inc. IntelliFont is a registered trademark of Agfa Corp. This document may also contain reference to other trademarks and registered trademarks for the various products listed which are believed to belong to the sources associated therewith.

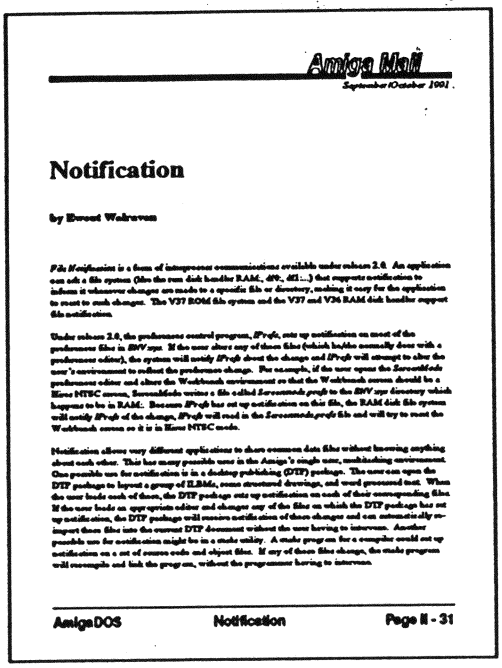
Amiga Mail
Volume II

Amiga Mail is published six times annually by the Commodore Applications and Technical Support Group (CATS). Its purpose is to support third-party developers of Amiga software and hardware worldwide by providing technical information and programming guidelines. Each issue features a collection of news and technical articles of special interest to Amiga product developers.

The Amiga Mail Reference Book (Volume II)

Collectively, the articles from this issue and past issues of Amiga Mail form a reference book. The book is broken up into chapters covering different aspects of Amiga programming (Exec, IFF, AmigaDOS...). All Amiga Mail issues are printed on three hole punch paper so they can easily be filed in a three ring reference book binder. To file the articles from this issue in the reference book, use the chapter and page numbers that appear at the foot of each page. Each page's chapter number is shown as a Roman numeral followed by its page number within that chapter.

For instance, the Amiga Mail article below "Notification", would be filed in chapter II of the binder, which is the "AmigaDOS" chapter. The article starts on page 31 of that chapter in the binder.


Amiga Mail
September/October 1991

Notification

by Everett Wedraven

File Notification is a form of interprocess communications available under release 2.0. An application can ask a file system (like the ram disk handler RAM:, df0:, df1:...) that supports notification to inform it whenever changes are made to a specific file or directory, making it easy for the application to react to such changes. The V37 ROM file system and the V37 and V36 RAM disk handler support file notification.

Under release 2.0, the preferences control program, *IPrefs*, sets up notification on most of the preference files in *ENV:* type. If the user alters any of these files (which he/she normally does with a preferences editor), the system will notify *IPrefs* about the change and *IPrefs* will attempt to alter the user's environment to reflect the preferences change. For example, if the user opens the *ScreenMode* preferences editor and alters the Workbench environment so that the Workbench screen should be a Hires NTSC screen, *ScreenMode* writes a file called *Sys:env/env.prefs* to the *ENV:* type directory which happens to be in RAM:. Because *IPrefs* has set up notification on this file, the RAM disk file system will notify *IPrefs* of the change, *IPrefs* will read in the *Sys:env/env.prefs* file and will try to reset the Workbench screen so it is in Hires NTSC mode.

Notification allows very different applications to share common data files without knowing anything about each other. This has many possible uses in the Amiga's multi-user, multitasking environment. One possible use for notification is in a desktop publishing (DTP) package. The user can open the DTP package to layout a group of ILBMs, create structured drawings, and word processed text. When the user loads each of these, the DTP package sets up notification on each of their corresponding files. If the user loads an appropriate editor and changes any of the files on which the DTP package has set up notification, the DTP package will receive notification of these changes and can automatically re-import these files into the current DTP document without the user having to intervene. Another possible use for notification might be in a make utility. A make program for a compiler could set up notifications on a set of source code and object files. If any of these files change, the make program will recompile and link the program, without the programmer having to intervene.

AmigaDOS Notification Page II - 31

Warning

The information contained herein is subject to change without notice. Commodore specifically does not make any endorsement or representation with respect to the use, results, or performance of the information (including without limitation its capabilities, appropriateness, reliability, currentness or availability).

Disclaimer

This information is provided "AS IS" without warranty of any kind, either express or implied. The entire risk as to the use of this information is assumed by the user. In no event will Commodore or its affiliated companies be liable for any damages, direct, indirect, incidental, special or consequential, resulting from any claim arising out of the information presented herein, even if it has been advised of the possibility of such damages. Some states do not allow the exclusion or limitation of such implied warranties, so the above limitations may not apply.

For more information on how to obtain a binder, chapter separators, and past issues of Amiga Mail, contact your local Commodore support manager, or, if you are in the U.S., contact CATS at (215) 431-9180.

From time to time, an issue of Amiga Mail is published with correction pages which replace existing pages in the Amiga Mail reference book. These appear at the end of an Amiga Mail issue after the Table of Contents update.

Each issue of Amiga Mail also comes with an update to the book's table of contents.

Amiga Mail Editor:
John Orr

Contributors:
Michael Sinz, Randell Jesup, Chris Green,
Peter Cherna, and Lauren Brown.

About Amiga Mail:
Amiga Mail is published on an Amiga 3000.
The software used to produce this issue includes: *PageStream 2.2*, New Horizon's *ProWrite 3.2.2*, the *SAS/C Amiga C Compiler*, Oxxi's *TurboText*, and Softwood, Inc.'s *Proper Grammar*.

Comments to:

John Orr
CATS
Commodore Business Machines, Inc.
1200 Wilson Drive
West Chester, PA 19380

or

via E-mail: johno@ cbmvax.commodore.com
[rutgers| uunet!] ! cbmvax ! johno

This publication is printed on recycled material.

Amiga Mail News

Commodore U.S.A. Enhances Product Line and Reduces List Prices

Commodore Business Machines, Inc. announced it has significantly reduced the U.S. manufacturers suggested retail price on selected models of its Amiga® series multimedia computers by up to 25%. In addition, the company introduced several new system configurations and the new A2386SX Bridgeboard™, which enables the Amiga 2000 and 3000 series computers to run MS-DOS® compatible software.

Commodore announced special promotional pricing to its dealers in March. The success of this promotional effort resulted in the company's decision to permanently adopt the lower prices, establishing new dealer and manufacturers suggested retail pricing (MSRP). One example of the substantial price reductions is the Amiga 2000 with 1084S color monitor, previously available at a MSRP of $1986. Its new MSRP is $1539, representing a 23% reduction.

The pricing moves are also part of Commodore's aggressive multimedia marketing campaign. This campaign is focused on the professional multimedia user involved in videographics, dynamic presentations, interactive training, or point-of-information kiosks. According to Commodore Vice President of Professional Sales Geoffrey Stilley, "We have long been recognized as a price performance leader in the multimedia marketplace and these new systems configurations and prices will keep us at the forefront. We believe the Amiga offers the most comprehensive and affordable solutions addressing the demanding needs of the multimedia professional."

One new system configuration specifically aimed at the video professional is an Amiga 2000-based system with a 100MB hard drive, Motorola® 68030 processor, 5MB of RAM and a 1084S monitor with an MSRP of $2699. In addition, the user can purchase the same system with an A3070 tape drive allowing for magnetic tape storage of 150MB of information at an MSRP of $3159.

Two additional configurations available through September 30, 1992 are an Amiga 3000-25/50 with a 50 MB hard drive, 2 MB of RAM and the new A2386SX Bridgeboard for an MSRP of $3399, and an Amiga 3000T-25/200 multimedia

IN THIS ISSUE

Optimized Window Refreshing
Martin Taillefer

Using Compugraphic Typefaces with Bullet
John Orr

Memory Limits for Various CDTV Configurations
Adam Levin-Delson

workstation with a 200MB hard drive, 5MB of RAM and a 1950 or 1960 VGA color monitor for an MSRP of $4499. Commodore director of Business Marketing David Archambault stated, "The A3000-25/50 with the new Bridgeboard is targeted at the multimedia professional who requires the video, audio, and animation capabilities of the Amiga, as well as the ability to run and communicate with an MS-DOS environment. Also, the A3000T-25/200 is ideal for the multimedia applications developer who requires lots of expandability and processing power at an affordable price."

Commodore's 386SX Bridgeboard give an Amiga system MS-DOS compatibility by allowing PC application software to run within a window under AmigaDOS™. It also allows the two systems to transfer files back and forth from the Amiga to the Bridgeboard. The new 386SX Bridgeboard is accompanied by MS-DOS 5.0 software and can be used with all Amiga 2000 and 3000 series computers. It has a processor clock speed of 20 MHz and will support a maximum of two floppy drives while sharing a hard drive with the Amiga side.

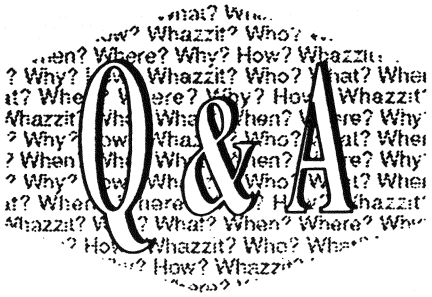
Commodore International Names Director of Applications and Technical Support

Commodore International announced the appointment of John F. Campbell as director of Commodore Applications and Technical Support (CATS), effective immediately. The CATS group is responsible for supporting and establishing development and quality standards for over 1500 developers who produce software and hardware products for the Amiga® and CDTV® Multimedia Player product lines. Campbell reports directly to Commodore International director of Worldwide Software Engineering, Ned McCook. "John is well prepared for his new role," said McCook. "His depth of product knowledge combined with his management skills and respect for our developers will help strengthen Commodore's relationship with our development community."

A graduate of the University of Pennsylvania with degrees in technology management and computer science, Campbell has held a number of key development positions with Commodore since joining in 1982. He assisted in the launch of the VIC-20® and Commodore 64® computers and is one of the people responsible for the tremendous software support they received from developers, which led to the success of both products.

Since 1985, Campbell has concentrated on bringing high quality, diverse products to market for the company's line of Amiga computers. A major highlight is AmigaVision™, for which he headed the team that defined the need for an authoring system that would take advantage of all the multimedia features of the Amiga. AmigaVision has won a number of industry awards and is broadly used by multimedia developers worldwide.

This position announcement coincides with moving the CATS group into Commodore's Engineering division. "This change will help strengthen ties between Commodore's development efforts and the work of our developers," noted Campbell. "We will now be in a better position to move our proprietary technology forward quickly so that Commodore can continue to provide leading edge hardware at affordable prices."



Q: The DOS Library function FGets() seems to read too many bytes if it doesn't come across a newline character. What's happening?

A: This is an excerpt from a recent Autodoc for FGets:

In V36 and V37, it copies one more byte than it should if it doesn't hit an EOF or newline. In the example above, it would copy 50 bytes and put a null in the 51st. This is fixed in dos V39. Workaround for V36/V37: pass in buffersize-1.

Q: Is there an interaction between DMA devices (such as hard drive controllers) and the 68040?

A: Yes. This interaction is only an issue when the 68040 is in copyback data cache mode and the timing is just right. Note that the chance of this interaction affecting any particular application is rather small. The interaction is as follows:

The 68040 data cache has a mode called copyback. In this mode, when the CPU modifies memory, it does not write the data to memory right away. It waits until the cache is needed to cache some other memory or someone explicitly flushes the cache. This creates a problem for DMA devices because they can read directly from memory without using the CPU. If a DMA device tries to read memory that happens to be in the CPUs data cache, the DMA device will read the wrong data.

To prevent problems with DMA, a DMA device has to call CachePreDMA() before accessing memory. This function does both address translations (for future MMU issues) and cache flushing as needed. When the transfer is complete, the device has to call CachePostDMA() which performs some cache flushes and whatever cleanup it needed.

Unfortunately, there is a nasty interaction between the copyback mode of the 68040 and flushed data areas.

In copyback mode, the 68040 uses complete cache lines (which are four long words in length) when copying data back and forth from the cache. There are good reasons for this, including that the CPU can do a full burst transfer (which is four long words). However, the 68040 does this even if only 2 of the long words in the cache line are changed. So, what does this mean?

Imagine a program that has two memory buffers right next to each other. The program is running on a 68040 in copyback mode. These buffers meet in the middle of a 128 bit boundary, so a cache line would overlap the end of the first buffer and the beginning of the second buffer. If the timing is just right, the following situation can occur:

The program calls CachePreDMA() and starts an asynchronous DMA write to the second buffer.

The program uses the CPU to write to the last two long words of the first buffer which causes the CPU to load the end of first buffer and the beginning of the second buffer as they are part of the same cache line.

The DMA write writes to the beginning of the second buffer and finishes the transfer.

The program calls CachePostDMA(), which causes the CPU to write the cache line that overlaps the first and second buffer.

Because the overlapping cache line still contains the data that was in the second buffer before the DMA transfer took place, the CPU overwrites the the first two long words of the second buffer.

The current version of *68040.library* (37.4) has the code needed to work around this problem. Note that if any developer used SetFunction() to patch some of the cache control functions, it is important to patch all of these functions with SetFunction() as needed to work around this problem. *68040.library* already does SetFunction() all of the cache functions to deal with copyback issues and this one just happens to be the most complex of them. Version 37.4 of the 68040.library is available through the CATS closed developer confoerences on BIX.

Q: I'm using ExAll() (dos.library 37.44) with ED_COMMENT. Everything works fine except that the string ead->ed_Comment points to isn't NULL-terminated.

Is this a bug or am I doing something wrong? If it's a bug, is there a more simple workaround than looking at the FileInfoBlock?

A: This is a bug. It turns out that the V37 FFS does the comment wrong when in ExAll(). It does it as a BSTR rather than a standard C-String. Other filesystems (such as RAM) do it right. Also, filesystems that do not directly support ExAll() but have DOS simulate ExAll() do it right, too.

Q: The 37.4 DOS library Autodoc says that the FPutC() function looks like this:

```
LONG FPutC(BPTR, UBYTE)
```

while version 37.4 of the <clib/dos_protos.h> include file looks like this:

```
LONG FPutC( BPTR fh, unsigned long ch)
```

Which one is right?

A: Would you believe neither? It's really:

```
LONG FPutC(BPTR, LONG ch)
```

There's no particularly good reason for it, other than that's how BCPL did it, and this is a trans-literated (directly translated) version of the BCPL. BCPL has only one real type: LONG.

It wouldn't make a difference except that the value returned for success is the longword passed in, and longword comparisons are done against things like '\n', etc. So you must pass in a longword value 0-255.

Q: How do I determine the frequency of the CIA B time of day (TOD) counter?

A: The CIA B time of day counter counts horizontal sync pulses. The counter rate varies depending on the video mode the Amiga is currently displaying. Instead of using the TOD counter, you want use one of the more conventioanl CIA timers, and scale the time constants based on whether the system is PAL or NTSC.

Table of Contents

July/August 1992

I
Programming Practices and Standards
general purpose Amiga programming

Tag Items and Tag Lists . . . . . I - 1
Staff

2.0 Version Strings . . . . . I - 3
Carolyn Scheppner

The ASL Library. . . . . . I - 7
John Orr

2.0 Compatibility Problem Areas . . . . . I - 19
Carolyn Scheppner, Bryce Nesbitt, Peter Cherna, and Darren Greenwald

Release 2.0 Specific Functions and Commands . . . . . I - 35
Compiled by Carolyn Scheppner

Callback Hooks . . . . . I - 43
David Junod

Debugging with Enforcer and Mungwall . . . . . I - 47
Carolyn Scheppner

II
AmigaDOS
dos.library

Executing External Programs with Release 2.0 . . . . . II - 1
Carolyn Scheppner

AmigaDOS Packet Interface Specification . . . . . II - 5
John Toebes

Standard Command Line Parsing . . . . . II - 27
John Orr

Notification . . . . . II - 31
Ewout Walraven

Using the AmigaDOS Pattern Matching Functions . . . . . II - 37
Ewout Walraven

The 2.0 Dos.library Path Name Handling Functions . . . . . II - 43
Ewout Walraven

Directory Scanning . . . . . II - 49
Ewout Walraven

Writing a UserShell . . . . . II - 65
Randell Jesup

III
Exec
the multitasking executive, exec.library

Using SetFunction() in a Debugger . . . . . III - 1
Ewout Walraven

68040 Compatibility Warning . . . . . III - 11
Michael Sinz

IV
Intuition and Workbench
the Amiga user interface

Obtaining an Icon Image . . . . . IV - 1
David Junod

An Introduction to V36 Screens and Windows . . . . . IV - 3
Ewout Walraven

Opening Screens and Windows on Any Amiga . . . . . IV - 17
Ewout Walraven

Introduction to Boopsi . . . . . IV - 23
John Orr

Writing a Boopsi Image Class . . . . . IV - 41
David N. Junod

Boopsi's Transparent Base Classes . . . . . IV - 57
Peter Cherna

AppWindows, AppIcons, and AppMenuItems . . . . . IV - 59
Fred Mitchell and John Orr

Making Your Windows More Shiny and Manageable . . . . . IV - 69
Ewout Walraven

Optimized Window Refreshing . . . . . IV - 77
Martin Taillefer

V
Graphics
animation, text, the Blitter, layers.library

Using the ECS Genlocking Features . . . . . V - 1
Ewout Walraven

Finding the Aspect Ratio . . . . . V - 11
Carolyn Scheppner

Amiga Font Scaling and Aspect Ratio . . . . . V - 15
John Orr

Using Compugraphic Typefaces with Bullet . . . . . V - 23
John Orr

VI
Sound and Music
audio, speech, MIDI

VII
IFF
iffparse.library and other IFF topics

The DR2D FORM . . . . . VII - 1
Ross Cunniff and John Orr

ECS Display Modes and ILBM CAMG . . . . . VII - 17
Carolyn Scheppner

Developing New IFF FORMs and Chunks . . . . . VII - 21
Christian Ludwig

VIII
Unix and Networking
Unix SVR4, NFS, TCP/IP, Ethernet

Using the Amiga A3000UX as a NFS File Server . . . . . VIII - 1
David Miller

Keeping Time--Interval Timing in Amiga UNIX . . . . . VIII - 9
David Miller

Developing Network Applications for the Amiga . . . . . VIII - 17
Dale Larson

A Shared Socket Library Server and Client . . . . . **VIII - 31**
John Wiederhirn and John Orr

SANA-II Network Device Specification . . . . . **VIII - 51**
Brian Jackson, Dale Larson, Greg Miller, and Kenneth Dyke

IX
Timer, Serial, and Parallel
timer.device, serial port, parallel port

Four Can Play–Supporting Parallel Port Joysticks . . . . . **IX - 1**
Christian Ludwig

X
Output
printers, printing, Postscript

XI
Input
user input, Commodities Exchange

Introduction to Commodities Exchange . . . . . **XI - 1**
John Orr

XII
CDTV
Commodore Dynamic Total Vision

Guidelines for CDTV Applications . . . . . **XII - 1**
Staff

Making CDTV Titles That Work on Amigas . . . . . **XII - 9**
Ben Phister

Memory Limits for Various CDTV Configurations . . . . . **XII - 17**
Adam Levin-Delson

XIII
Hardware
Cards, chips, connectors

Video Adapter Diagrams . . . . . **XIII - 1**
Adam Levin-Delson

Basic Input/Output

The Basic Input/Output actions are supported by both handlers and file systems. In this way, the application can get a stream level access to both devices and files. One difference that arises between the two is that a handler will not necessarily support an ACTION_SEEK while it is generally expected for a file system to do so.

These actions work based on a FileHandle which is filled in by one of the three forms of opens:

```
ACTION_FINDINPUT          1005    Open(..., MODE_OLDFILE)
ACTION_FINDOUTPUT         1006    Open(..., MODE_NEWFILE)
ACTION_FINDUPDATE         1004    Open(..., MODE_READWRITE)
ARG1:  BPTR FileHandle to fill in
ARG2:  LOCK Lock on directory that ARG3 is relative to
ARG3:  BSTR Name of file to be opened (relative to ARG1)

RES1:  BOOL Success/Failure (DOSTRUE/DOSFALSE)
RES2:  CODE Failure code if RES1 is DOSFALSE
```

All three actions use the lock (ARG2) as a base directory location from which to open the file. If this lock is NULL, then the file name (ARG3) is relative to the root of the current volume. Because of this, file names are not limited to a single file name but instead can include a volume name (followed by a colon) and multiple slashes allowing the file system to fully resolve the name. This eliminates the need for AmigaDOS or the application to parse names before sending them to the file system. Note that the lock in ARG2 must be associated with the file system in question. It is illegal to use a lock from another file system.

The calling program owns the file handle (ARG1). The program must initialize the file handle before trying to open anything (in the case of a call to Open(), AmigaDOS allocates the file handle automatically and then frees it in Close()). All fields must be zero except the fh_Pos and fh_End fields which should be set to -1. The Open() function fills in the fh_Type field with a pointer to the MsgPort of the handler process. Lastly, the handler must initialize fh_Arg1 with something that allows the handler to uniquely locate the object being opened (normally a file). This value is implementation specific. This field is passed to the READ/WRITE/SEEK/ END/TRUNCATE operations and not the file handle itself.

FINDINPUT and FINDUPDATE are similar in that they only succeed if the file already exists. FINDINPUT will open with a shared lock while FINDUPDATE will open it with a shared lock but if the file doesn't exist, FINDUPDATE will create the file. FINDOUTPUT will always open the file (deleting any existing one) with an exclusive lock.

```
ACTION_READ               'R'     Read(...)
ARG1:  ARG1 fh_Arg1 field of the opened FileHandle
ARG2:  APTR Buffer to put data into
ARG3:  LONG Number of bytes to read

RES1:  LONG Number of bytes read.
        0 indicates EOF.
       -1 indicates ERROR
RES2:  CODE Failure code if RES1 is -1
```

This action extracts data from the file (or input channel) at the current position. If fewer bytes

remain in the file than requested, only those bytes remaining will be returned with the number of bytes stored in RES1. The handler indicates an error is indicated by placing a -1 in RES1 and the error code in RES2. If the read fails, the current file position remains unchanged. Note that a handler may return a smaller number of bytes than requested, even if not at the end of a file. This happens with interactive type file handles which may return one line at a time as the user hits return, for example the console handler, CON:.

```
ACTION_WRITE                'W'     Write(...)
ARG1:  ARG1 fh_Arg1 field of the opened file handle
ARG2:  APTR Buffer to write to the file handle
ARG3:  LONG Number of bytes to write

RES1:  LONG Number of bytes written.
RES2:  CODE Failure code if RES1 not the same as ARG3
```

This action copies data into the file (or output channel) at the current position. The file is automatically extended if the write passes the end of the file. The handler indicates failure by returning a byte count in RES1 that differs from the number of bytes requested in ARG3. In the case of a failure, the handler does not update the current file position (although the file may have been extended and some data overwritten) so that an application can safely retry the operation.

```
ACTION_SEEK                 1008    Seek(...)
ARG1:  ARG1    fh_Arg1 field of the opened FileHandle
ARG2:  LONG    New Position
ARG3:  LONG    Mode:   OFFSET_BEGINNING,OFFSET_END, or  OFFSET_CURRENT

RES1:  LONG    Old Position.   -1 indicates an error
RES2:  CODE    Failure code if RES1 = -1
```

This packet sets the current file position. The new position (ARG2) is relative to either the beginning of the file (OFFSET_BEGINNING), the end of the file (OFFSET_END), or the current file position (OFFSET_CURRENT), depending on the mode set in ARG3. Note that ARG2 can be negative. The handler returns the previous file position in RES1. Any attempt to seek past the end of the file will result in an error and will leave the current file position in an unknown location.

```
ACTION_END                  1007    Close(...)
ARG1:   ARG1    fh_Arg1 field of the opened FileHandle

RES1:   LONG    DOSTRUE
```

This packet closes an open file handle. This function generally returns a DOSTRUE as there is little the application can do to recover from a file closing failure. If an error is returned under 2.0, DOS will not deallocate the file handle. Under 1.3, it does not check the result.

```
ACTION_SET_PROTECT        21      SetProtection(...)
ARG1:   Unused
ARG2:   LOCK    Lock to which ARG3 is relative
ARG3:   BSTR    Name of object (relative to ARG2)
ARG4:   LONG    Mask of new protection bits

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = DOSFALSE
```

This action allows an application to modify the protection bits of an object. The 4 lowest order bits (RWED) are a bit peculiar. If their respective bit is set, that operation is not allowed (i.e. if a file's delete bit is set the file is *not* deleteable). By default, files are created with the RWED bits set and all others cleared. Additionally, any action which modifies a file is required to clear the A (archive) bit. See the *dos/dos.h* include file for the definitions of the bit fields.

```
ACTION_SET_COMMENT        28      SetComment(...)
ARG1:   Unused
ARG2:   LOCK    Lock to which ARG3 is relative
ARG3:   BSTR    Name of object (relative to ARG2)
ARG4:   BSTR    New Comment string

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = DOSFALSE
```

This action allows an application to set the comment string of an object. If the object does not exist then DOSFALSE will be returned in RES1 with the failure code in RES2. The comment string is limited to 79 characters.

```
ACTION_SET_DATE           34      SetFileDate(...) in 2.0
ARG1:   Unused
ARG2:   LOCK    Lock to which ARG3 is relative
ARG3:   BSTR    Name of Object (relative to ARG2)
ARG4:   CPTR    DateStamp

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = DOSFALSE
```

This action allows an application to set an object's creation date.

2.0 only

```
ACTION_FH_FROM_LOCK       1026    OpenFromLock(lock)
ARG1:   BPTR    BPTR to file handle to fill in
ARG2:   LOCK    Lock of file to open

RES1:   BOOL    Success/failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 = NULL
```

This action open a file from a given lock. If this action is successful, the file system will essentially steal the lock so a program should not use it anymore. If ACTION_FH_FROM_LOCK fails, the lock is still usable by an application.

2.0 only

```
ACTION_SAME_LOCK          40      SameLock(lock1,lock2)
ARG1:   BPTR    Lock 1 to compare
ARG2:   BPTR    Lock 2 to compare

RES1:   LONG    Result of comparison, one of
  DOSTRUE           if locks are for the same object
  DOSFALSE          if locks are on different objects
RES2:   CODE    Failure code if RES1 is LOCK_DIFFERENT
```

This action compares the targets of two locks. If they point to the same object, ACTION_SAME_LOCK should return LOCK_SAME.

2.0 only

```
ACTION_MAKE_LINK          1021    MakeLink(name,targ,mode)
ARG1:    BPTR     Lock on directory ARG2 is relative to
ARG2:    BSTR     Name of the link to be created (relative to ARG1)
ARG3:    BPTR     Lock on target object or name (for soft links).
ARG4:    LONG     Mode of link, either LINK_SOFT or LINK_HARD

RES1:    BOOL     Success/Failure (DOSTRUE/DOSFALSE)
RES2:    CODE     Failure code if RES1 is DOSFALSE
```

This packet causes the file system to create a link to an already existing file or directory. There are two kinds of links, hard links and soft links. The basic difference between them is that a file system resolves a hard link itself, while the file system passes a string back to DOS telling it where to find a soft linked file or directory. To the packet level programmer, there is essentially no difference between referencing a file by its original name or by its hard link name. In the case of a hard link, ARG3 is a lock on the file or directory that the link is "linked" to, while in a soft link, ARG3 is a pointer (CPTR) to a C-style string.

In an over-simplified model of the ROM file system, when asked to locate a file, the system scans a disk looking for a file header with a specific (file) name. That file header points to the actual file data somewhere on the disk. With hard links, more than one file header can point to the same file data, so data can be referenced by more than one name. When the user tries to delete a hard link to a file, the system first checks to see if there are any other hard links to the file. If there are, only the hard link is deleted, the actual file data the hard link used to reference remains, so the existing hard links can still use it. In the case where the original link (not a hard or soft link) to a file is deleted, the file system will make one of its hard links the new "real" link to the file. Hard links can exist on directories as well. Because hard links "link" directly to the underlying media, hard links in one file system cannot reference objects in another file system.

Soft links are resolved through DOS calls. When the file system scans a disk for a file or directory name and finds that the name is a soft link, it returns an error code (ERROR_IS_SOFT_LINK). If this happens, the application must ask the file system to tell it what the link the link refers to by calling ACTION_READ_LINK. Soft Links are stored on the media, but instead of pointing directly to data on the disk, a soft link contains a path to its object. This path can be relative to the lock in ARG1, relative to the volume (where the string will be prepended by a colon ':'), or an absolute path. An absolute path contains the name of another volume, so a soft link can reference files and directories on other disks.

2.0 only

```
ACTION_READ_LINK          1024  ReadLink(port,lck,nam,buf,len)
ARG1:    BPTR     Lock on directory that ARG2 is relative to
ARG2:    CPTR     Path and name of link (relative to ARG1).  NOTE: This is a C
string not a BSTR
ARG3:    APTR     Buffer for new path string
ARG4:    LONG     Size of buffer in bytes

RES1:    LONG     Actual length of returned string, -2 if there isn't enough
space in buffer,or -1 for other errors
RES2:    CODE     Failure code
```

This action reads a link and returns a path name to the link's object. The link's name (plus any necessary path) is passed as a CPTR (ARG2) which points to a C-style string, *not a BSTR*. ACTION_READ_LINK returns the path name in ARG3. The length of the target string is returned in RES1 (or a -1 indicating an error).

```
BOOL = MatchFunc(hookptr, data,typeptr)
                  A0      A1    A2
hookptr    Pointer to hook being called
data       Pointer to (partially) filled in ExAllData for item being checked.
typeptr    Pointer to longword indicating the type of the ExAll request (ARG4).
```

This function is expected to return DOSTRUE if the entry is accepted and DOSFALSE if it is to be discarded.

2.0 only

```
ACTION_EXAMINE_FH        1034    ExamineFH(fh,fib)
ARG1:   BPTR    File handle on open file
ARG2:   BPTR    FileInfoBlock to fill in

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

This function examines a file handle and fills in the FileInfoBlock (found in ARG2) with information about the current state of the file. This routine is analogous to the ACTION_EXAMINE_OBJECT action for locks. Because it is not always possible to provide an accurate file size (for example when buffers have not been flushed or two processes are writing to a file), the fib_Size field (see *dos/dos.h*) may be inaccurate.

2.0 only

```
ACTION_ADD_NOTIFY        4097    StartNotify(NotifyRequest)
ARG1:   BPTR    NotifyRequest structure

RES1:   BOOL    Success/Failure (DOSTRUE/DOSFALSE)
RES2:   CODE    Failure code if RES1 is DOSFALSE
```

This action asks a file system to notify the calling program if a particular file is altered. A file system notifies a program either by sending a message or by signaling a task.

```
struct NotifyRequest {
        UBYTE *nr_Name;
        UBYTE *nr_FullName;             /* set by dos - don't touch */
        ULONG nr_UserData;              /* for applications use */
        ULONG nr_Flags;

        union {

            struct {
                struct MsgPort *nr_Port;        /* for SEND_MESSAGE */
            } nr_Msg;

            struct {
                struct Task *nr_Task;           /* for SEND_SIGNAL */
                UBYTE nr_SignalNum;             /* for SEND_SIGNAL */
                UBYTE nr_pad[3];
            } nr_Signal;
        } nr_stuff;

        ULONG nr_Reserved[4];           /* leave 0 for now */

        /* internal use by handlers */
        ULONG nr_MsgCount;              /* # of outstanding msgs */
        struct MsgPort *nr_Handler;     /* handler sent to (for EndNotify) */
};
```

To use this packet, an application needs to allocate and initialize a NotifyRequest structure (see above). As of this writing, NotifyRequest structures are not allocated by AllocDosObject(), but this may change in the future. The handler gets the watched file's name from the nr_FullName field. The current file system does not currently support wild cards in this field, although there is nothing to prevent other handlers from doing so.

The string in nr_FullName must be an absolute path, including the name of the root volume (no assigns). The absolute path is necessary because the file or its parent directories do not have to exist when the notification is set up. This allows notification on files in directories that do not yet exist. Notification will not occur until the directories and file are created.

An application that uses the StartNotify() DOS call does *not* fill in the NotifyRequest's nr_FullName field, but instead fills in the nr_Name field. StartNotify() takes the name from the nr_Name field and uses GetDeviceProc() and NameFromLock() to expand any assigns (such as ENV:), storing the result in nr_FullName. Any application utilizing the packet level interface instead of StartNotify() must expand their own assigns. Handlers must not count on nr_Name being correct.

The notification type depends on which bit is set in the NotifyRequest.nr_Flags field. If the NRF_SEND_MESSAGE bit is set, an application receives notification of changes to the file through a message (see NotifyMessage from *dos/notify.h*). In this case, the nr_Port field must point to the message port that will receive the notifying message . If the nr_Flags NRF_SEND_SIGNAL bit is set, the file system will signal a task instead of sending a message. In this case, nr_Task points to the task and nr_SignalNum is the signal number. *Only one of these two bits should be set!*

When an application wants to limit the number of NotifyMessages an handler can send per NotifyRequest, the application sets the NRF_WAIT_REPLY bit in the nr_Flags field. This bit tells the handler not to send new NotifyMessages to a NotifyRequest's message port if the application has not returned a previous NotifyMessage. This pertains only to a specific NotifyRequest--if other NotifyRequests exist on the same file (or directory) the handler will still send NotifyMessages to the other NotifyRequest's message ports. The NRF_WAIT_REPLY bit only applies to message notification.

If an application needs to know if a file or directory exists at the time the application sets up notification on that file or directory, the application can set the NRF_NOTIFY_INITIAL bit in the nr_Flags field. If the file or directory exists, the handler sends an initial message or gives an initial signal.

Handlers should only perform a notification when the actual contents of the file have changed. This includes ACTION_WRITE, ACTION_SET_DATE, ACTION_DELETE, ACTION_RENAME_OBJECT, ACTION_FINDUPDATE, ACTION_FINDINPUT, and ACTION_FINDOUTPUT. It may also include other actions such as ACTION_SET_COMMENT or ACTION_SET_PROTECT, but this is not required (and may not be expected by the application as there is no need to reread the data).

2.0 only

```
ACTION_REMOVE_NOTIFY      4098     EndNotify(NotifyRequest)
ARG1:    BPTR     Pointer to previously added notify request

RES1:    BOOL     Success/Failure (DOSTRUE/DOSFALSE)
RES2:    CODE     Failure code if RES1 is DOSFALSE
```

This action cancels a notification (see ACTION_ADD_NOTIFY) . ARG1 is the NotifyRequest structure used to initiate the notification. The handler should abandon any pending notification messages. Note that it is possible for a file system to receive a reply from a previously sent notification message even after the notification has been terminated. It should accept these messages silently and throw them away.

Amiga Mail
Volume II
July/August 1992

Optimized Window Refreshing

by Martin Taillefer

Maintaining the graphical contents of an Amiga window can be difficult. There are many subtleties associated with the process known as window refreshing. At present, many applications refresh their windows in suboptimal ways, or fail to refresh correctly under all conditions. This article attempts to explore and resolve the window refreshing problems commonly encountered by applications.

Damaging Information

Although many people think of Intuition as the Amiga's windowing system, the lower-level Layers library actually handles most of the work involved in maintaining Intuition's windowed environment. The Layers library divides a single physical display (a BitMap) into multiple virtual displays. Each of these virtual displays is known as a layer. Intuition uses the functions in the Layers library to move, resize, and depth arrange these layers.

Normally, each Intuition window consists of a single layer. Intuition adds borders and gadgetry to the layer to give it the familiar Intuition window appearance. Intuition also takes care of monitoring user input to let the user move, resize, and depth arrange windows. When Intuition wishes to change the size or position of a window, it calls functions in the Layers library to do most of the grunt work.

One of the main reasons for a layered rendering system is to provide multiple independent logical rendering regions to applications. The layer is the basis for the Intuition Window. The functions in the Layers library allow several applications to render into the same physical display (an Intuition Screen for example) without worrying about interfering with each other. As far as the application is concerned, it has an entire display all to itself.

There is a limit to the isolation that this layered environment offers applications. Because layers can overlap, by moving, resizing, or deleting a layer, a program can uncover portions of underlying layers. These newly exposed portions are called "damage regions". A layer can sustain damage when a task performs layers operations on it or other layers in the same display. When the Layers library damages a layer, it sets the layer's LAYERREFRESH bit in the Flags field of the Layer structure.

When damage occurs to a layer, the damaged region of the layer must be repaired by redrawing it. The entity responsible for redrawing the damaged region depends on the type of layer. The Layers library offers three types of layers: simple refresh, smart refresh, and superbitmap. Subsequently, Intuition bases its three types of windows on these three types of layers. The difference between each type has to do with the way in which each handles damage repair.

When a layers operation damages a simple refresh layer, the entire burden of repairing the layer's damage rests on the application that created the layer (if the application is using Intuition windows, which are built on top of layers, Intuition shares the burden of damage repair with the application. More on this later). This is because a simple refresh layer does not preserve its contents. The advantage of this type of layer is that it doesn't use much memory. The disadvantage is that every time a layers operation reveals a portion of a simple refresh layer, the application must explicitly rerender the exposed damaged regions.

Smart refresh layers help the application by doing some of the refreshing automatically. If a layer operation conceals a portion of a layer, that portion is automatically copied to an off-screen buffer. If a layer operation later reveals that portion of the layer, the Layers library uses the temporary buffer to update the revealed region. The Layers library does not leave the LAYERREFRESH bit set in this case because the Layers library took care of the damage. The application still needs to refresh the smart refresh layer whenever it is made larger, as the Layers library has no idea what should be in the newly exposed areas. In this case, the Layers library will leave the LAYERREFRESH bit set.

Finally, superbitmap layers totally eliminate the need for an application to refresh the display. The Layers library maintains a complete off-screen buffer representing the layer's contents. When a layer operation exposes new portions of a layer, the Layers library automatically updates these regions by copying from the off-screen buffer. The Layers library will never leave the LAYERREFRESH bit set for a superbitmap layer.

For the application programmer, a superbitmap layer offers the simplest approach to refreshing the layer as the entire burden of repairing damaged regions falls on the Layers library. This added convenience does have a cost--because of its off-screen buffer, the superbitmap layer requires a significantly larger amount of memory compared to the simple and smart refresh layers. The Layers library allocates this memory even if the layer never sustains damage.

When To Refresh

The way in which an application determines that it needs to refresh a layer depends on whether an application deals with layers directly via the Layers library, or indirectly via Intuition. It is important to keep in mind that each Intuition window has a Layer at its core. If an application uses Intuition windows it may not use the Layers library to create, delete, move, resize, or update the window layers. The application must use the corresponding Intuition functions instead.

If an application uses layers directly, it must look at a layer's LAYERREFRESH bit to tell if the layer sustained damage and needs repair. Since the application created and maintains its layers, it knows when damage can occur, so it checks for damage at those times.

Intuition manages an arbitrary number of layers on behalf of any number of client applications. All windowing operations happen on Intuition's time frame. Since other applications and the user can ask Intuition to manipulate window layers around at any time, it is not possible for a window-based application to know by itself when to refresh its window. That would involve polling the window layer's LAYERREFRESH bit, which is a big no-no in a multitasking system. Instead, Intuition provides a refresh notification mechanism through the IDCMP system.

Whenever Intuition performs a layer operation that can damage a window layer, it checks the damage state of each layer in the current screen by inspecting each layer's LAYERREFRESH bit. If Intuition finds that a layer's LAYERREFRESH bit is set, Intuition takes care of refreshing the damaged areas of the window that are Intuition's responsibility (for example, the window borders and gadgets). Following that, Intuition looks at the WFLG_NOCAREREFRESH bit in the Window structure. If this bit is set, Intuition's refresh processing for the window is complete, and all damage region information is discarded. However, If the bit is clear, Intuition sends an IDCMP_REFRESHWINDOW message to the window's IDCMP port, essentially asking the window to refresh itself.

So, instead of having to poll the LAYERREFRESH bit, an application can just wait for Intuition to tell it to refresh its window. This is convenient and fits in well with the IDCMP mechanism.

Scrolling Your Life Away

A scrolling display is a part of many user interfaces. An easy way to make people think an application is slow is to give it a sluggish user interface. Regardless of the actual speed at which an application performs its job, if its user interface management is slow, the user will get the impression the whole application is slow.

The ScrollRaster() and ClipBlit() routines are the two principal ways of scrolling data within a layer. ScrollRaster() moves the data within a RastPort. ClipBlit() moves data from one RastPort to the next, but works equally well if the source and destination RastPorts are the same.

Smart refresh layers are relatively easy to scroll with either ScrollRaster() or ClipBlit(). Although ClipBlit() has a theoretical advantage over ScrollRaster(), the performance of these functions is generally equivalent. ScrollRaster() does a minimum of two Blitter operations: one to scroll the data, and a generally smaller one to erase the area that was scrolled away. ClipBlit() can do a single blit to move the data in the layer, and leaves the rest of the display alone. Thus, while scrolling large amounts of data, an application can get better CPU/Blitter processing overlap by using ClipBlit(). Consider the following pseudo-code of what happens in ScrollRaster() versus ClipBlit():

Within the ScrollRaster() function:

Wait for any outstanding Blitter operation to complete
Initiate a Blitter operation to move the data in the window
Wait for the Blitter operation to complete
Initiate a second small Blitter operation to clear the scrolled region
Return to the caller while the small blit is still in progress

Within the ClipBlit() function:

Wait for any outstanding Blitter operation to complete
Initiate a Blitter operation to move the data in the window
Return to the caller while the blit is still in progress

Since the Blitter and the CPU can run concurrently, the second approach provides the most throughput, as the Blitter performs most of the scrolling work *at the same time* the application code runs after ClipBlit(). In practice though, ScrollRaster() and ClipBlit() run at about the same speed.

For a smart refresh window, ScrollRaster() and ClipBlit() do not present a problem. As long as the application only scrolls the contents of the window and not the window borders, the Layers library will take care of fixing any damage. Intuition will never know what happened. For a simple refresh window however, there is a problem. The reason has to do with the following:

```
+--------------+
|aaaaaaaaaaaaaa|
|bbbb+----+bbbb|
|cccc|    |cccc|
|dddd+----+dddd|
|eeeeeeeeeeeeee|
+--------------+
```

Figure 1a

```
+--------------+
|bbbbaaaaaabbbb|
|cccc+----+cccc|
|dddd|    |dddd|
|eeee+----+eeee|
|              |
+--------------+
```

Figure 1b

Figure 1a represents a small window that the user placed on top of a larger simple refresh window. When an application scrolls the larger window's contents upwards with ScrollRaster(), the result looks like Figure 1b. This operation exposes two portions of the larger window, the section at the bottom of the large window (the part ScrollRaster() clears) and also the area above the smaller window (which ScrollRaster() leaves intact).

The application that owns the larger window has to refresh the two portions exposed by ScrollRaster(). The application knows to refresh the bottom section of the window because the damage is a direct result of the application calling ScrollRaster(). The problem is the area above the smaller window. The application does not know that there is a window overlapping its larger window, so it does not directly know about any damage resulting from other overlapping layers. Because the application used a graphics.library function to manipulate the window, Intuition does not know about the damage either. The result is that the larger window is damaged and no one knows it.

If the larger window had been a smart refresh window, the Layers library would have cached this portion of the larger window and ScrollRaster() would have taken action to make sure the cached region got restored. For a simple refresh window, this portion only gets added to the window layer's damage list and no rendering actually occurs.

The solution to this problem is for the application to act as Intuition would if it were scrolling. When Intuition manipulates a layer, it checks the LAYERREFRESH bit of each of its layers to see if any of them was damaged. After the application calls ScrollRaster() on its window, the application has to look at the LAYERREFRESH bit of the window's layer. If the bit is set, damage exists and the application needs to repair the window. As this window is the only window that can sustain layer damage as a result of the call to ScrollRaster(), the application needs to check only its own window's layer for damage.

Note that to scroll the contents of a simple refresh window, an application has to use ScrollRaster() rather than ClipBlit(). The reason is ClipBlit() does not add anything to the window layer's damage region, so an application will never know about the damage. Also, unlike ScrollRaster(), ClipBlit() does not scroll the window layer's damage region. If the layer already has damage when an application calls ScrollRaster(), the position of the damage region will move along with the data in the layer. Note that prior to Release 2, ScrollRaster() did not scroll the layer's damage list.

Faster Rendering

When rendering to a RastPort, an application can improve its rendering performance if it renders using a limited number of colors. Each RastPort contains a mask field which specifies the writeable bitplanes of the RastPort's BitMap. If a RastPort's bitplane is write-protected, the system ignores that bitplane when rendering to the RastPort.

An application can control the BitMap mask by using the SetWrMsk() macro (defined in *<graphics/gfxmacros.h>*). This macro accepts a pointer to a RastPort, and a new mask. The mask is an 8 bit value. Each bit of the value represents a BitMap bitplane. For example, a mask of value 5, which is 00000101 in binary, restricts rendering in that BitMap to planes 0 and 2 which are the only two bits set in the binary value.

One type of application that can improve its performance by using the RastPort Mask is the text editor. Typically, a text editor needs to render its text in a single color, generally color 1. This means the editor only needs to render to bitplane 0. All other planes will always remain blank. If these planes are going to remain blank, why should the editor bother to render to them or scroll them?

When scrolling large sections of the display, ScrollRaster() can make the display look unstable because it continuously clears portions of the display. ClipBlit() eliminates this visual nuisance as it does not clear the display. Unfortunately, ClipBlit() by itself is useless for scrolling a simple refresh window because it does not deal with damage regions. The solution is sneaky but quite effective. An application can do the following each time it scrolls a simple refresh layer:

Set the RastPort Mask to limit the number of writeable bitplanes.
Use ClipBlit() to scroll the data.
Set the RastPort Mask to 0.
Use ScrollRaster() to scroll the same data.

In the procedure above, ClipBlit() scrolls the window contents but not the damage regions. ScrollRaster() scrolls the damage region, but because the RastPort mask is 0, ScrollRaster() does not affect the window contents. Setting the RastPort mask to 0 prevents the system from disturbing the data in any of the planes of the BitMap.

The above trick can come in quite handy. It is fairly fast as well, although it can involve some hidden overhead. Even though ScrollRaster() doesn't move data on the display, it still needs to go through all the layer's clipping regions, which can be time consuming.

Using Multiple RastPorts

A RastPort specifies attributes needed to perform many rendering operations. These include the RastPort's foreground, background, and drawing mode. The Graphics library functions SetAPen(), SetBPen(), and SetDrMd() set each of these attributes, respectively.

Although these functions appear to have fairly simple purposes in life, the SetAPen(), SetBPen(), and SetDrMd() are quite CPU intensive routines. These functions require recalculating values that the OS caches in private parts of the RastPort. If an application only requires a few different combinations of foreground/background/draw mode, it can improve its performance by using a different RastPort structure for each combination. An application sets the attributes of several RastPorts only once which is generally more efficient that setting the attributes of one RastPort every time the rendering attributes change.

Assume an application has a window in which it renders all of its data in pen 1, and clears any part of its display using color 0. Such an application can improve its rendering performance by doing the following:

```
struct RastPort dataRP;
struct RastPort clrRP;

dataRP = *window->RPort;
SetAPen(&dataRP,1);
SetBPen(&dataRP,0);

clrRP = *window->RPort;
SetAPen(&clrRP,0);

/* renders to the window's RastPort in color 1 */
Text(&dataRP,"hello",5);

/* clears a section of the window's RastPort in color 0 */
RectFill(&clrRPort,0,0,10,10);
```

Refreshing a Sizable Window

A sizable window is by far the trickiest to refresh correctly. The most important and often overlooked point is that an application needs to ensure the size of its window does not change while it is refreshing the window. If the application doesn't and the user changes the window size, the application will refresh the window at the window's old size rather than its new size. This can severely corrupt the appearance of the window.

There are two main ways an application can keep the size of its window stable while refreshing the window. The first method is to keep the size of the window locked most of the time, unlocking the window only when the user tries to size the window. The other approach is to lock the window size only while rendering to the window.

The first method is part of Intuition's IDCMP mechanism. To lock a window's size, an application sets the window's `IDCMP_SIZEVERIFY` IDCMP flag. When the user attempts to size the window by clicking on the window's sizing gadget, Intuition notifies the application by sending an `IDCMP_SIZEVERIFY` message to the window's IDCMP port. Intuition will keep the window's size locked until the application returns the `IDCMP_SIZEVERIFY` message.

In general this scheme works well, but it does have two problems. First, if the application is busy doing some processing, such as recalculating a spreadsheet, it may not notice that the message arrived until it is done with its current processing. The result is the user will not be able to size the window until the application is finished processing, which might more time than the user wants.

The second problem occurs when the application is waiting for input from Intuition, such as in the middle of an EasyRequest() call. If the user clicks on the sizing gadget of the application's window while the requester is up, Intuition will wait for the system will enter a deadlock. Intuition will wait for the application to send back will not see the IDCMP_SIZEVERIFY event until the user satisfies the EasyRequest(). The result will be a system deadlock. Many applications suffer from this problem.

The second problem occurs when the application is waiting for input from Intuition, such as in the middle of an EasyRequest() call. If the user clicks on the sizing gadget of the application's window while the requester is up, the system will enter a deadlock. When the user clicks the sizing gadget, Intuition sends the IDCMP_SIZEVERIFY message and waits for a reply. Because the application is already waiting for the EasyRequest() to return, the application cannot send back the reply. Many applications suffer from this problem.

As of Release 2, Intuition has adapted and avoids these deadlocks. Intuition will time out the sizing operation if the application does not process the IDCMP_SIZEVERIFY message within a given time period. Although the user can no longer deadlock the system, this situation can still confuse the user because clicking on the window's sizing gadget no longer sizes the window. That does not mean an application should rely upon Intuition to avoid the deadlock. An application should always avoid these conditions.

In general, it is simpler and safer for the application to lock a window only during the rendering process. The application can do this by surrounding all rendering operations with calls to the Layers library functions LockLayer() and UnlockLayer(). LockLayer() locks the size and position of a layer. While a window's layer is locked, an application can safely look at the window's current size and render to it without any danger of the size changing. Once the application finishes rendering, it unlocks the window's layer by calling UnlockLayer(). When using this method, an application must not set the window's IDCMP_SIZEVERIFY bit. Be careful which system functions an application calls while it has a layer locked. Only use the Graphics library rendering functions and the simple Intuition rendering functions (i.e., PrintIText(), DrawImage(), etc.). In particular, avoid calls that deal with gadgets (including RefreshGList()) and other locks (i.e., LockIBase() and LockLayerInfo()).

BeginRefresh() and EndRefresh()

To improve the performance of repairing damage to simple and smart refresh windows, Intuition can put a window into a special refresh state using Intuition's BeginRefresh() function:

```
VOID BeginRefresh(struct Window *window)
```

While a window is in this state, the system restricts attempts to render into the window to the window's damaged regions. Because the system ignores rendering operations outside the window's damage regions, an application only refreshes the parts of a window that need refreshing. This can significant decrease the amount of time necessary to refresh the window. This also reduces the possibility of visual flicker that can happen if an application has to redraw the entire contents of a window.

To end a window's special refresh state, use EndRefresh():

```
VOID EndRefresh(struct Window *window, BOOL Complete)
```

Compared to BeginRefresh(), EndRefresh() has an extra parameter, a boolean value. If this value is TRUE, Intuition assumes that all of the refreshing for this window is finished and removes all of the window layer's damaged regions. In most cases, this value should be TRUE. See the Autodoc for BeginRefresh() and EndRefresh() for more information.

Backfill Hook

Whenever a layer sustains damage, the damaged region is cleared to color 0. Clearing the region requires an often lengthy Blitter operation which is unnecessary if the application plans to redraw the damaged region anyway. Such an application can improve its performance by preventing the Layers library from clearing the damaged regions. This is what layer backfill hooks are for.

Backfill hooks are a new Layers library feature added in Release 2. A backfill hook is a custom function that an application attaches to a specific layer. Whenever damage occurs to a layer, the Layers library calls that layer's custom backfill hook. The Layers library passes the position and dimensions of the damage area to the backfill hook. The backfill hook can render into the damage area, refreshing it. Theoretically, the backfill hook can redraw a layer's damaged regions instead of clearing the damage area to color 0.

Unfortunately, refreshing a layer from a backfill hook can be quite difficult. The backfill hook code is usually called from a different task than the main application. If the backfill hook and the application do not properly arbitrate access to the application's data, dangerous race conditions can occur if the hook and the application try to access the same data. Another problem with rendering through the

backfill hook is that no clipping is available. The backfill hook must do its own clipping to ensure that no rendering goes outside the dimensions specified when the hook is called, which in itself can be quite difficult.

For many situations, the best use of a backfill hook is to have it do nothing. Whenever the hook is called, it simply returns without doing any rendering. This has the effect of eliminating the extra blit done to clear the damage region to color 0. This can speed layer operations quite a bit.

There is one problem with a no-op backfill hook. If the application is busy doing some processing and damage occurs to its layer, the display will remain dirty until the application finishes its processing and notices that the layer is damaged. This damage can include remnants of system imagery, like window borders, which can confuse the user.

One way to overcome this problem is to use a backfill hook that changes its behavior depending on the state of the application. If the application is not too busy to notice damage to its window's layer, the backfill hook does not erase the display. However, if the application is too busy to refresh its damaged window, the backfill hook clears the damaged portion of the display. Using this method, the backfill hook will not waste time erasing the damage when the application will update it immediately, but the backfill hook will erase the damage if the application can't refresh for a while.

```
;/* Optimrefresh.c - Execute me to compile me with SAS/C 5.10a
LC -cfistqmcu -v -y -j73 optimrefresh.c
Blink FROM LIB:c.o,optimrefresh.o TO optimrefresh LIBRARY LIB:LC.lib,LIB:Amiga.lib
quit ;*/

/* This program demonstrates optimal window refreshing using a scrolling text
 * display as a sample.
 *
 * Copyright   1992 Martin Taillefer
 * All Rights Reserved
 */

#include <exec/types.h>
#include <exec/libraries.h>
#include <exec/memory.h>
#include <utility/hooks.h>
#include <utility/tagitem.h>
#include <graphics/gfxmacros.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>
#include <intuition/gadgetclass.h>
#include <dos.h>

#include <clib/exec_protos.h>
#include <clib/intuition_protos.h>
#include <clib/graphics_protos.h>
#include <clib/layers_protos.h>
#include <clib/alib_protos.h>
#include <clib/dos_protos.h>

/*****************************************************************************/


/* There is one Line structure for every line of text in our fictional
 * document.
 */
struct Line
{
    struct MinNode ln_Link;     /* to link the lines together          */
    STRPTR         ln_Text;     /* pointer to the text for this line   */
    ULONG          ln_TextLen;  /* the length of the text for this line */
};


/*****************************************************************************/


/* system libraries */
struct Library  *IntuitionBase;
struct Library  *GfxBase;
struct Library  *LayersBase;

/* global display handles */
struct Screen   *screen;
struct Window   *window;
struct Gadget   *scroller;
struct Hook      refreshHook;

struct RastPort render;
struct RastPort clear;

/* our document along with associated view information */
struct MinList   document;
ULONG            numLines;
ULONG            topLine;
ULONG            oldTopLine;
ULONG            linesVisible;
ULONG            columnsVisible;
ULONG            fontHeight;
ULONG            fontWidth;
ULONG            viewHeight;
ULONG            viewWidth;
ULONG            usefulWidth;
ULONG            usefulHeight;

/* a state flag indicating whether the main application is busy */
BOOL             taskBusy;
```

```
/*****************************************************************************/

VOID InitDocument(VOID);
VOID FreeDocument(VOID);
VOID EventLoop(VOID);
VOID __asm BackFillHook(register __a2 struct RastPort    *rp,
                        register __a1 struct BackFillMsg *bfm);

/*****************************************************************************/


/* This is where it all begins.
 */
ULONG main(void)
{
    /* open the system libraries we need.
     */
    IntuitionBase = OpenLibrary("intuition.library",37);
    GfxBase       = OpenLibrary("graphics.library",37);
    LayersBase    = OpenLibrary("layers.library",37);

    if (IntuitionBase && GfxBase && LayersBase)
    {
        /* get a pointer to the default public screen */
        if (screen = LockPubScreen(NULL))
        {
            /* allocate and initialize a scroller as a BOOPSI object */
            if (scroller = NewObject(NULL,"propgclass",
                  GA_RelRight,     -13,
                  GA_Top,          1+screen->WBorTop+screen->Font->ta_YSize+1,
                  GA_Width,        10,
                  GA_RelHeight,    -12-(screen->WBorTop+screen->Font->ta_YSize+1),
                  GA_RelVerify,    TRUE,
                  GA_Immediate,    TRUE,
                  GA_FollowMouse,  TRUE,
                  GA_RightBorder,  TRUE,
                  PGA_Borderless,  TRUE,
                  PGA_Freedom,     FREEVERT,
                  PGA_Total,       1,
                  PGA_Visible,     1,
                  PGA_Top,         0,
                  PGA_NewLook,     TRUE,
                  TAG_DONE))
            {
                /* initialize data used by the backfill hook */
                refreshHook.h_Entry = ( ULONG (*)() )BackFillHook;  /* point the */
                taskBusy            = TRUE;                 /* hook to our routine. */

                /* open the window */
                if (window = OpenWindowTags(NULL,
                      WA_Left,          0,
                      WA_Top,           0,
                      WA_PubScreen,     screen,
                      WA_AutoAdjust,    TRUE,
                      WA_CloseGadget,   TRUE,
                      WA_DepthGadget,   TRUE,
                      WA_DragBar,       TRUE,
                      WA_SizeGadget,    TRUE,
                      WA_SizeBRight,    TRUE,
                      WA_Title,         "Optimized Refresh Sample",
                      WA_SimpleRefresh, TRUE,
                      WA_Activate,      TRUE,
                      WA_Gadgets,       scroller,
                      WA_MinWidth,      32,
                      WA_MinHeight,     10+12+(screen->Font->ta_YSize+1),
                      WA_MaxWidth,      -1,
                      WA_MaxHeight,     -1,
                      WA_IDCMP,         IDCMP_CLOSEWINDOW | IDCMP_NEWSIZE
                                        | IDCMP_REFRESHWINDOW | IDCMP_GADGETUP
                                        | IDCMP_GADGETDOWN | IDCMP_MOUSEMOVE
                                        | IDCMP_VANILLAKEY,
                      WA_BackFill,      &refreshHook,
                      TAG_DONE))
                {
                    /* initialize our document structure */
                    InitDocument();
```

```
                    /* process user events in the window */
                    EventLoop();

                    /* free our document structure */
                    FreeDocument();

                    /* close up shop */
                    CloseWindow(window);
                }
                /* free the scroller BOOPSI object */
                DisposeObject(scroller);
            }
            /* unlock the default public screem */
            UnlockPubScreen(NULL,screen);
        }
    }

    /* close the libraries we opened */
    CloseLibrary(LayersBase);
    CloseLibrary(GfxBase);
    CloseLibrary(IntuitionBase);

    /* tell the shell everything is all right */
    return(0);
}


/*****************************************************************************/


/* This function initializes our document. That means allocating 100
 * Line structures and linking them together in an Exec list. The lines
 * are filled with a pattern of text so we have something to display
 * in our window
 */
VOID InitDocument(VOID)
{
struct Line *line;
UWORD        i,j;

    NewList((struct List *)&document);
    numLines = 0;
    i        = 100;
    while (i--)
    {
        if (line = AllocVec(sizeof(struct Line)+91,MEMF_CLEAR|MEMF_PUBLIC))
        {
            line->ln_Text    = (STRPTR)((ULONG)line + sizeof(struct Line));
            line->ln_TextLen = 40;
            AddTail((struct List *)&document,(struct Node *)line);
            numLines++;

            j = 0;
            while (j < 90)
            {
                line->ln_Text[j] = (numLines % 96) + 32;
                j++;
            }
        }
    }
}


/*****************************************************************************/


/* This function frees all the memory allocated by InitDocument() */
VOID FreeDocument(VOID)
{
struct Line *line;

    while (line = (struct Line *)RemHead((struct List *)&document))
        FreeVec(line);
}


/*****************************************************************************/
```

```
/* This is the message packet passed by layers.library to a backfill hook.
 * It contains a pointer to the layer that has been damaged, a Rectangle
 * structure that defines the bounds of the damage. No rendering can occur
 * outside of these coordinates.
 *
 * The backfill hook is also passed a RastPort in which the rendering
 * should be performed.
 */
struct BackFillMsg
{
    struct Layer     *bf_Layer;
    struct Rectangle  bf_Bounds;
    LONG              bf_OffsetX;
    LONG              bf_OffsetY;
};


VOID __asm BackFillHook(register __a2 struct RastPort    *rp,
                        register __a1 struct BackFillMsg *bfm)
{
struct RastPort crp;

    crp       = *rp;            /* copy the rastport                        */
    crp.Layer = NULL;           /* eliminate bogus clipping from our copy */

    if (taskBusy)
    {
        SetWrMsk(&crp,0xff);    /* if the main task is busy, clear all planes */
    }
    else
    {
        SetWrMsk(&crp,0xfe);    /* otherwise, clear all planes except plane 0 */
    }

    SetAPen(&crp,0);                    /* set the pen to color 0          */
    SetDrMd(&crp,JAM2);                 /* set the rendering mode we need */
    RectFill(&crp,bfm->bf_Bounds.MinX,  /* clear the whole area            */
                  bfm->bf_Bounds.MinY,
                  bfm->bf_Bounds.MaxX,
                  bfm->bf_Bounds.MaxY);
}


/*****************************************************************************/


/* Adjust the scroller object to reflect the current window size and
 * scroll offset within our document
 */
VOID SetScroller(struct Window *window, struct Gadget *scroller,
                 ULONG linesVisible, ULONG numLines, ULONG topLines)
{
    SetGadgetAttrs(scroller,window,NULL,PGA_Visible, linesVisible,
                                        PGA_Total,   numLines,
                                        PGA_Top,     topLine,
                                        TAG_DONE);
}


/*****************************************************************************/


/* Render a single line of text at a given position */
VOID RenderLine(UWORD x, UWORD y, UWORD w, STRPTR text, ULONG len)
{
    Move(&render,x,y);                  /* move the cursor to the position */

    if (len > columnsVisible)           /* is line is longer than allowed? */
        len = columnsVisible;           /* yes, so reduce its length       */

    Text(&render,text,len);             /* write to the window             */

    if (len < columnsVisible)
        RectFill(&clear,render.cp_x,y-render.TxBaseline,
                        x+w-1,y-render.TxBaseline+fontHeight-1);

}
```

```
/*****************************************************************************/

/* This function performs most of the rendering work needed by our sample.
 * It first locks the window's layer to insure it doesn't get sized during
 * the rendering process. It then looks at the current window size and
 * adjusts its rendering variables in consequence. If the damage parameter
 * is set to TRUE, the routine then proceeds to explicitly erase any area
 * of the display to which we will not be rendering in the rendering loop.
 * This erases any left over characters that could be left if the user sizes
 * the window smaller. Finally, the routine determines which lines of the
 * display need to be updated and goes on to do it.
 */
VOID RefreshView(BOOL damage)
{
ULONG        i;
struct Line *line;
UWORD        x,y;

    /* lock the window's layer so its size will not change */
    LockLayer(NULL,window->WLayer);

    /* determine various values based on the current size of the window */
    viewWidth      = window->Width - window->BorderLeft - window->BorderRight;
    fontWidth      = window->RPort->Font->tf_XSize;
    columnsVisible = viewWidth / fontWidth;

    viewHeight     = window->Height - window->BorderTop - window->BorderBottom;
    fontHeight     = window->RPort->Font->tf_YSize;
    linesVisible   = viewHeight / fontHeight;

    usefulWidth = columnsVisible * fontWidth;

    if (linesVisible > numLines)
    {
        usefulHeight = numLines * fontHeight;
        topLine      = 0;
    }
    else if (topLine + linesVisible > numLines)
    {
        topLine      = (numLines - linesVisible);
        usefulHeight = (numLines - topLine) * fontHeight;
    }
    else
    {
        usefulHeight = linesVisible * fontHeight;
    }

    /* if we were called because of damage, we must erase any left over
     * garbage
     */
    if (damage)
    {
        /* erase anything left over on the right side of the window */
        if ((window->BorderLeft + usefulWidth < window->Width - window->BorderRight)
        &&   usefulHeight)
        {
            RectFill(&clear,window->BorderLeft + usefulWidth,
                            window->BorderTop,
                            window->Width - window->BorderRight - 1,
                            window->BorderTop + usefulHeight - 1);
        }

        /* erase anything left over on the bottom of the window */
        if ((window->BorderLeft < window->Width - window->BorderRight)
        &&  (window->BorderTop + usefulHeight < window->Height - window->BorderBottom))
        {
            RectFill(&clear,window->BorderLeft,
                            window->BorderTop + usefulHeight,
                            window->Width - window->BorderRight - 1,
                            window->Height - window->BorderBottom - 1);
        }
    }

    /* if we have at least one line and one column to render... */
    if (usefulHeight && usefulWidth)
    {
        /* get a pointer to the first line currently visible */
        line = (struct Line *)document.mlh_Head;
        i    = topLine;
        while (line->ln_Link.mln_Succ && i--)
            line = (struct Line *)line->ln_Link.mln_Succ;

        if (damage
        || (topLine >= oldTopLine + linesVisible - 1)
        || ((oldTopLine > linesVisible)
        && (topLine <= oldTopLine - linesVisible + 1)))
        {
            /* the whole display must be redrawn */
            x = window->BorderLeft;
            y = window->BorderTop + window->RPort->Font->tf_Baseline;
            i = linesVisible;
        }
        else if (topLine < oldTopLine)
        {
            /* we just need to scroll the text */
            ScrollRaster(&render,0,-(LONG)((oldTopLine - topLine) * fontHeight),
                         window->BorderLeft,
                         window->BorderTop,
                         window->BorderLeft+usefulWidth-1,
                         window->BorderTop+usefulHeight-1);

            /* indicates what section needs to be redrawn */
            x = window->BorderLeft;
            y = window->BorderTop + window->RPort->Font->tf_Baseline;
            i = oldTopLine - topLine;
        }
        else if (topLine > oldTopLine)
        {
            /* we just need to scroll the text */
            ScrollRaster(&render,0,(topLine - oldTopLine) * fontHeight,
                         window->BorderLeft,
                         window->BorderTop,
                         window->BorderLeft+usefulWidth-1,
                         window->BorderTop+usefulHeight-1);

            /* indicates what section needs to be redrawn */
            i = linesVisible - (topLine - oldTopLine);
            while (line->ln_Link.mln_Succ && i--)
                line = (struct Line *)line->ln_Link.mln_Succ;

            x = window->BorderLeft;
            y = window->BorderTop + window->RPort->Font->tf_Baseline
                + (fontHeight * (linesVisible - (topLine - oldTopLine)));
            i = topLine - oldTopLine;
        }
        else
        {
            /* we don't need to render anything */
            i = 0;
        }

        /* render all the lines we need */
        while (i-- && line->ln_Link.mln_Succ)
        {
            RenderLine(x,y,usefulWidth,line->ln_Text,line->ln_TextLen);
            y    += fontHeight;
            line  = (struct Line *)line->ln_Link.mln_Succ;
        }
    }

    /* unlock the layer so normal operations can resume */
    UnlockLayer(window->WLayer);

    /* keep track of what the current top line is. That way, when we
     * come back in this routine later, and *topLine* has changed, we
     * can tell how many lines we need to scroll in order to sync up the
     * display
     */
    oldTopLine = topLine;
}

/*****************************************************************************/
```

```
/* Whenever the application is busy, this function is called. It will
 * change the behavior of the backfill hook in order to improve the
 * appearance of the display until the application completes its lengthy
 * task.
 *
 * You could also set a busy pointer in the document window in this routine
 * to tell the user you are not listening to him for awhile.
 */
VOID BusyState(BOOL makeBusy)
{
    taskBusy = makeBusy;

    if (LAYERREFRESH & window->WLayer->Flags)
    {
        BeginRefresh(window);
        RefreshView(TRUE);
        EndRefresh(window,TRUE);
    }
}


/*****************************************************************************/


/* This routine is a typical event processor. It looks and acts on all events
 * arriving at the window's port.
 */
VOID EventLoop(VOID)
{
struct IntuiMessage *intuiMsg;
ULONG               class;

    topLine    = 0;
    oldTopLine = 0;

    /* initialize rendering attributes we are going to use */
    render = *window->RPort;
    SetDrMd(&render,JAM2);
    SetWrMsk(&render,1);        /* we only want to render in the first plane */
    SetAPen(&render,1);

    /* initialize clearing attributes we are going to use */
    clear = *window->RPort;
    SetDrMd(&clear,JAM2);
    SetWrMsk(&clear,1);         /* we only want to clear the first plane */
    SetAPen(&clear,0);

    /* render the initial display */
    RefreshView(TRUE);

    /* set the initial scroller position and size */
    SetScroller(window,scroller,linesVisible,numLines,topLine);

    /* we aren't busy, so register that fact */
    BusyState(FALSE);

    while (TRUE)
    {
        /* if the LAYERREFRESH flag is set in the window's layer, it
         * means the layer has some damage we should repair.
         */
        if (LAYERREFRESH & window->WLayer->Flags)
        {
            /* enter optimized repair state */
            BeginRefresh(window);

            /* redraw the whole display through the optimized repair
             * region
             */
            RefreshView(TRUE);

            /* tell the system we are done repairing the window
             */
            EndRefresh(window,TRUE);
        }

        /* nothing left to do but wait for user input */
        WaitPort(window->UserPort);
        intuiMsg = (struct IntuiMessage *)GetMsg(window->UserPort);
        class    = intuiMsg->Class;
        ReplyMsg(intuiMsg);

        /* we got a message, so act on it */
        switch (class)
        {
            /* user clicked on the close gadget, exit the program */
            case IDCMP_CLOSEWINDOW  : return;

            /* user sized the window. We need to redraw the whole
             * display in order to eliminate any garbage. Start by
             * calling BeginRefresh() and EndRefresh() to eliminate
             * the window's damage regions then completely redraw
             * the window contents.
             */
            case IDCMP_NEWSIZE      : BeginRefresh(window);
                                      EndRefresh(window,TRUE);
                                      RefreshView(TRUE);
                                      SetScroller(window,
                                                  scroller,
                                                  linesVisible,
                                                  numLines,
                                                  topLine);
                                      break;

            /* Intuition is telling us damage occured to our layer.
             * Don't bother doing anything, the check at the top of the
             * loop will catch this fact and refresh the display
             *
             * Even though we don't do anything with these events, we
             * still need them to be sent to us so we will wake up and
             * look at the LAYERREFRESH bit.
             */
            case IDCMP_REFRESHWINDOW: break;

            /* user is playing with the scroller. Get the scroller's current
             * top line and synchronize the display to match it
             */
            case IDCMP_GADGETUP     :
            case IDCMP_GADGETDOWN   :
            case IDCMP_MOUSEMOVE    : GetAttr(PGA_Top,scroller,&topLine);
                                      RefreshView(FALSE);
                                      break;

            /* whenever a key is hit, we fake becoming busy for 4
             * seconds. During that time, try to size and depth arrange
             * the window to see what happens to its contents
             */
            case IDCMP_VANILLAKEY   : BusyState(TRUE);
                                      Delay(200);
                                      BusyState(FALSE);
                                      break;
        }
    }
}
```

Amiga Mail
Volume II
July/August 1992

Using Compugraphic Typefaces with Bullet

by John Orr

One of the improvements made to the Amiga's operating system for Workbench Release 2.1 is programmatic control of AGFA's IntelliFont® scaling engine. With this engine, application programs can fully utilize Compugraphic (CG) typefaces installed by *Fountain* (the CG typeface install that comes with 2.04 and 2.1). Some of the features that the font scaling engine offers include:

- ❑ Rasterization of a typeface to arbitrary vertical and horizontal resolutions.
- ❑ Baseline rotation of glyphs to an arbitrary angle.
- ❑ Glyph shearing (italicizing) to any angle from -45 to 45 degrees (inclusive).
- ❑ Access to kerning tables.
- ❑ Algorithmic emboldening.

Starting the Engine

There are several steps involved in using a font outline on the Amiga.

1. Open the font contents file (the ".font" file) and verify that it has a corresponding outline tag file (an ".otag" file).

All system supported fonts on the Amiga have a font contents file. From this file, an application can determine a font's type, so the application knows how to utilize the font. The font contents file is a FontContentsHeader structure (defined in *<diskfont/diskfont.h>*). The first field of that structure (fch_FileID) contains an ID identifying the font's type. If that type is OTAG_ID (0x0F03), the font is an outline and it should have a corresponding otag file. The otag file should be in the same directory as the font contents file.

2. Open the otag file, verify that it is valid, load its tag list into memory, and resolve any memory indirections in the tag list.

The otag file contains a tag list that describes the typeface. All of these tags are defined in *<diskfont/diskfonttag.h>* as either level 1, level 2, or level 3 outline tags. Level 1 tags are required to be present in an otag file. Level 2 and 3 tags are optional. See the include file for more information on the tag levels.

The first tag of the otag file is always `OT_FileIdent`. Its value is the size of the otag file. This tag is here to verify the validity of the otag file. If the first tag is not `OT_FileIdent`, or the `OT_FileIdent` tag value is not the size of the otag file, the otag file is invalid, so don't attempt to use it. If the file is valid, copy the entire file into a buffer.

The tags from the otag file have a special `OT_Indirect` bit. If this bit is set, the tag's value is an indirect reference to data defined elsewhere in the otag file. The tag's value is the offset to the data (in bytes) from the beginning of the otag file. For example, the otag file *fonts:CGTimes.otag* that is on the 2.04 Release disks contains the tag `OT_Family` (0x80009003), which has its `OT_Indirect` bit set. The value of the `OT_Family` tag is 195, meaning that the data for it--the NULL terminated string "CG Times"--is located 195 bytes into the otag file.

Of course, if an application read the file *fonts:CGTimes.otag* into a memory buffer, the "CG Times" string would be 195 bytes from the beginning of the buffer. The `OT_Family` tag must point to the absolute address of its data, so when an application loads an otag file into memory, it has to resolve the indirection of the `OT_Indirect` tags in memory. The application can do this by adding the buffer address to each `OT_Indirect` tag value.

3. Find out the name of the typeface's scaling engine and obtain a pointer to the engine's GlyphEngine structure.

One of the level 1 outline tags is the `OT_Engine` tag. This tag refers to the name of this typeface's scaling engine. At present there is only one scaling engine available on the Amiga. It is named *Bullet*. This is the IntelliFont scaling engine. The name is left over from the original implementation of the IntelliFont engine used on the Amiga. The scaling engine itself is in its own Exec library, called *bullet.library*. To open the engine, build a complete library name by adding the string ".library" to the `OT_Engine` string, and open it with OpenLibrary(). Don't assume that `OT_Engine` will always be the string "bullet". In the future, Commodore or some third party developer may create additional scaling engines libraries that will allow the Amiga to use other types of outline typefaces (PostScript, Nimbus-Q, etc.). Using the proper library name will help ensure compatibility with future possible scaling engines.

All scaling engine libraries contain several functions:

OpenEngine()	If successful, returns a pointer to the library's GlyphEngine structure.
CloseEngine()	Releases the GlyphEngine structure obtained in OpenEngine().
SetInfo()/SetInfoA()	Sets current parameters of a scaling engine (the current typeface, the current point size, the current output resolution, etc.)
ObtainInfo()/ObtainInfoA()	Queries a scaling engine for glyph information (a glyph's bitmap, the kerning value between two glyphs, etc.).
ReleaseInfo()/ReleaseInfoA()	Releases data obtained with ObtainInfo()/ObtainInfoA().

To obtain a pointer to a GlyphEngine structure for a particular scaling library, use that library's OpenEngine() routine. The function takes no arguments.

4. Tell the engine which typeface to use.

Setting a scaling engine's current typeface involves the SetInfoA() (or SetInfo()) function and the Level 0 tags from *<diskfont/diskfonttag.h>*. The SetInfoA() function takes two parameters, a pointer to the GlyphEngine structure, and a tag list of level 0 outline tags. The Level 0 tags act as commands for a scaling engine, some of which are for setting scaling engine parameters (with SetInfo() or SetInfoA()), and some of which are for querying information from a scaling engine (with ObtainInfo() or ObtainInfoA()).

Two tags set a scaling engine's current typeface: OT_OTagPath and OT_TagList. The OT_OTagPath tag points to the full path name of a typeface's otag file (for example, *fonts:CGTimes.otag*). The OT_OTagList tag points to the tag list created in step 2 above.

5. Set other scaling engine parameters.

There are three other parameters the scaling engine needs in order answer queries for information:

```
OT_DeviceDPI          OT_PointHeight          OT_GlyphCode
```

The `OT_DeviceDPI` tag refers to the resolution of the output device. The tag value's high word is the horizontal resolution and the low word is the vertical resolution. Both are unsigned words measured in dots per inch.

The `OT_PointHeight` tag refers to the height of a typeface in points. One point is approximately equal to 1/72 of an inch (AGFA/Compugraphic defines the point to be 0.01383 inches). For CG typefaces, this height is the distance between baselines from one line to the next (without any leading adjustments). The point height is represented as a fixed point, two's complement binary number, with the point situated in the middle of a long word. This means the lower word is the fractional portion and the upper word the whole number portion (the number covers the powers of two from 2^{15} through 2^{-16}).

For those who may not know, a two's complement number is a way of representing negative numbers. To find the two's complement of a negative number, find the one's complement of the absolute value of that number (change all the binary ones to zeros and all the zeros to ones) and then add one to the result. To change from two's complement, subtract one from the two's complement number and find the one's complement of the number. For example,

Before conversion to two's complement, the absolute value of the one byte decimal quantity -32 is represented as:

```
In binary         0010 0000  ($20)
One's complement  1101 1111  ($DF)
Add One           0000 0001  ($01)
-----------------------------------
Two's complement  1110 0000  ($E0)
```

So the number -32 is encoded as 1110 0000 or 0xE0 in hex. Notice that the high bit (the sign bit) of the encoded number is set if the number is negative. If the number is zero or greater, the high bit is clear. This procedure is independent of where the "point" is in the negative number (the "point" in this case is the divider between the whole portion of the number and the fractional portion). When the computer adds one to the one's complement, it does not consider where the "point" is in the one's complement, the computer just treats the one's complement value as a whole integer. For example, to encode the decimal quantity -531/256 as a two byte, fixed point, two's complement binary number where the point is situated in the middle of the two bytes:

```
531/256 = -(2 + 19/256) = -2.13 in hex

In binary           0000 0010.0001 0011   ($02.13 in hex, with the point)

One's complement    1111 1101 1110 1100   ($FDEC  in hex ignore the point from now on)
Add One             0000 0000 0000 0001   ($0001)
-----------------------------------------
Two's complement    1111 1101 1110 1101   ($FDED)
```

Notice that the one added to the one's complement is in the 2^{-8} (= 1/256) column.

The OT_GlyphCode tag refers to the current glyph. When an application asks the scaling engine to rasterize a glyph, this is the glyph the engine renders. The scaling engine uses Unicode encoding to represent glyphs. Unicode is an international character encoding standard that encompasses many of the world's national scripts in a 16-bit code space. Conveniently, the Amiga's Latin-1, 8-bit character set corresponds to the same glyphs as the Unicode standard. To set the current glyph to a character from the Amiga character set, use the same Latin-1 code, but pad it out to a 16-bit value.

Because the Compugraphic typefaces use their own character set, the scaling engine in the bullet.library has to map the Unicode glyph codes to Compugraphic glyph codes. Note that the Unicode standard encompasses many more glyphs than just the Latin-1 character set or the Compugraphic character set, so many of the characters in the Unicode set do not map to any glyphs in the Compugraphic set. For example, Unicode includes several Asian ideogram sets, that the Compugraphic set does not. The result is the vast majority of the Unicode characters are not available using Compugraphic typefaces. The Compugraphic character set covers roughly 400 glyphs. For more information on the UniCode standard, see the Unicode Consortium's book *The Unicode Standard, Worldwide Character Encoding* published by Addison-Wesley (ISBN 0-201-56788-1).

Rasterizing a Glyph

Once an application has set up the scaling engine, obtaining a glyph is a matter of asking for it. As was mentioned earlier, the ObtainInfoA()/ObtainInfo() function queries a scaling engine for glyph information. This function accepts a pointer to a GlyphEngine structure and a tag list. To ask for a rasterization of a glyph, pass ObtainInfo() the OT_GlyphMap tag. The engine expects the OT_GlyphMap value to be an address where it can place the address of a GlyphMap structure. The GlyphMap structure (defined in *<diskfont/glyph.h>*) is for reading only and looks like this:

```
struct GlyphMap {
    UWORD  glm_BMModulo;    /* # of bytes in row: always multiple of 4 */
    UWORD  glm_BMRows;      /* # of rows in bitmap */
    UWORD  glm_BlackLeft;   /* # of blank pixel columns at left of glyph */
    UWORD  glm_BlackTop;    /* # of blank rows at top of glyph */
    UWORD  glm_BlackWidth;  /* span of non-blank columns (horizontal span of the glyph) */
    UWORD  glm_BlackHeight; /* span of non-blank rows (vertical span of the glyph) */
    FIXED  glm_XOrigin;     /* distance from upper left corner of bitmap to lower */
    FIXED  glm_YOrigin;     /*   left of glyph (baseline), in fractional pixels */
    WORD   glm_X0;          /* approximation of XOrigin in whole pixels */
    WORD   glm_Y0;          /* approximation of YOrigin in whole pixels */
    WORD   glm_X1;          /* approximation of XOrigin + Width */
    WORD   glm_Y1;          /* approximation of YOrigin + Width */
    FIXED  glm_Width;       /* character advance, as fraction of em width */
    UBYTE *glm_BitMap;      /* actual glyph bitmap */
};
```

The glm_BitMap field points to a single bitplane bitmap of the glyph. This bitmap is not necessarily in Chip RAM, so if an application needs to use the blitter to render the glyph, it has to copy the bitmap to a Chip RAM buffer. The fields glm_BMModulo and glm_BMRows are the dimensions of the whole bitmap. The glyph itself does not occupy the entire bitmap area. The fields glm_BlackLeft, glm_BlackTop, glm_BlackWidth, and glm_BlackHeight describe the position and dimension of the glyph within the bitmap. The fields glm_XOrigin and glm_YOrigin are the X and Y offsets from the bitmap's upper left corner to the glyph's lower left corner. The lower left corner lies on the glyph's baseline. These X and Y offsets are in fractional pixels. The fields glm_X0 and glm_Y0 are rounded versions of glm_XOrigin and glm_YOrigin, respectively.

The glm_Width field is a measure of the width of the glyph in fractions of an em (pronounced like the letter 'M'). This value is a fixed point binary fraction. The em is a relative measurement. A distance of one em is equal to the point size of the typeface. For example, in a 36 point typeface, one em equals 36 points which is approximately equal to a half inch. For a 72 point typeface, one em equals 72 points which is approximately equal to one inch.

When an application is finished with the GlyphMap structure, it must use the ReleaseInfoA() or ReleaseInfo() function to relinquish the GlyphMap structure. This function uses the same format as ObtainInfoA()/ObtainInfo(), except the data value of the `OT_GlyphMap` tag is a pointer to the GlyphMap structure (rather than a pointer to a pointer).

Kerning

The IntelliFont scaling engine also supports two types of kerning. One type of kerning is called text kerning, which is for regular bodies of text. The other type of kerning is called design kerning, which is for more obvious displays, like headlines. The basic difference is that design kerning is generally more tightly spaced than text kerning.

Before asking for a kerning pair, an application has to tell the engine which character pair to kern. It does this using one of the SetInfo() functions to set the primary glyph, `OT_GlyphCode`, and the secondary glyph code, `OT_GlyphCode2`.

To ask the scaling engine for a kerning value, use one of the ObtainInfo() functions with the `OT_TextKernPair` (for text kerning) or `OT_DesignKernPair` (for design kerning) tags. The engine expects the value of the kerning tag to be an address where it can store a four byte long kerning value. The kerning value is a fixed point binary fraction of an em square (like the glm_Width field from the GlyphMap structure). This value is the distance to remove from the primary character advance (the glm_Width of `OT_GlyphCode`) when rendering the secondary glyph (`OT_GlyphCode2`) immediately following the primary glyph.

Unlike other ObtainInfo() tags, the scaling engine does not allocate any resources when answering queries about kerning values. Applications do not have to use ReleaseInfo() functions for kerning queries made with either `OT_TextKernPair` or `OT_DesignKernPair`.

Width Lists

An application can find the widths of a typeface's glyphs using the OT_WidthList tag with one of the ObtainInfo() functions. The engine expects the OT_WidthList value to be an address where it can place the address of a MinList structure. This MinList points to a list of GlyphWidthEntry structures. The GlyphWidthEntry structure (defined in *<diskfont/glyph.h>*) is for reading only and looks like this:

```
struct GlyphWidthEntry {
    struct MinNode gwe_Node;  /* on list returned by OT_WidthList inquiry */
    UWORD          gwe_Code;  /* entry's character code value */
    FIXED          gwe_Width; /* character advance, as fraction of em width */
};
```

The MinList contains an entry for each valid Unicode glyph ranging from the primary glyph, OT_GlyphCode, through the secondary glyph, OT_GlyphCode2. The engine does not create a GlyphWidthEntry structure for codes without glyphs (for example the codes before the space character in the Latin-1 character set).

When an application is finished with the width list, it must use one of the ReleaseInfo() functions to relinquish the list. This function uses the same format as the ObtainInfo() functions, except the data value of the OT_WidthList tag is a pointer to the MinList (rather than a pointer to a pointer). The primary and secondary code values do not have to remain constant while an application is using a width list. The engine deallocates the width list resources independently of the primary and secondary code values, so these can change after obtaining a width list.

The following code fragment asks the scaling engine, ge, for a list of character widths of the Unicode glyphs ranging from unicode (OT_GlyphCode) to unicode2 (OT_GlyphCode2), inclusive. The fragment steps through the list of widths, printing each one.

```
struct MinList *widthlist;
struct GlyphWidthEntry *widthentry;
. . .

if (SetInfo(ge,
            OT_GlyphCode,   unicode,
            OT_GlyphCode2,  unicode2,
            TAG_END) == OTERR_Success)
{
    if (ObtainInfo(ge, OT_WidthList, &widthlist, TAG_END) == OTERR_Success)
    {
        for (widthentry = (struct GlyphWidthEntry *) widthlist->mlh_Head;
            widthentry->gwe_Node.mln_Succ;
            widthentry = (struct GlyphWidthEntry *)
            widthentry->gwe_Node.mln_Succ)
        {
            printf("$%lx: %ld.%ld\n",
                widthentry->gwe_Code,
                widthentry->gwe_Width>>16,
                ((widthentry->gwe_Width&0xffff)*10000)>>16);
        }
        ReleaseInfo(ge, OT_WidthList, widthlist, TAG_END);
    }
}
. . .
```

Notice that the ObtainInfo() functions (as well as the SetInfo() functions) return an error code (the error codes are defined in *<diskfont/oterrors.h>*). If that error code is equal to OTERR_Success, the operation was successful.

Rotating

Taking advantage of other features of the Bullet library is a matter of setting other engine parameters using one of the SetInfo() functions with some other level 0 tags. One interesting feature of the IntelliFont engine is its ability to rotate glyphs. By setting the OT_RotateSin and OT_RotateCos values, the IntelliFont engine can rotate a glyph's baseline from the glyph origin (the glm_XOrigin and glm_YOrigin from the GlyphMap structure).

The OT_RotateSin and OT_RotateCos are, respectively, the sine and cosine of the baseline rotation angle. The engine can rotate to any angle. *The sine and cosine must correspond to the same angle and must be in the sine and cosine value range (0 through 1 inclusive).* The engine does not do any error checking on the sine and cosine values. As a result, the engine will yield strange results if the sine and cosine are from very different angles or if these values are out of range for sines and cosines (greater than 1). By default, the engine sets these values to 0.0 and 1.0, the sine and cosine of 0 degrees. These values are encoded as fixed point binary fractions (the negative values are two's complement).

When setting the baseline rotation, an application must set both the sine and cosine. It must set OT_RotateSin first, then OT_RotateCos. An application can set both values in the same SetInfo() function, but the sine must come first. For example, to set the rotation angle to 150 degrees:

The sine of 150 degrees is 0.5 which is 0x00008000 in hex. The cosine of 150 degrees is approximately -0.866 which is approximately 0xffff224c in hex (two's complement), so:

```
. . .

if (SetInfo(ge,
            OT_RotateSin, 0x8000,
            OT_RotateCos, 0xFFFF224C,
            TAG_END) == OTERR_Success) /* If SetInfo() returns OTERR_Success, */
                                       /* it worked OK. */
{   /* The baseline rotation has been set, now the application can render it. */

    . . .
```

Shearing

Like baseline rotation, glyph shearing (also known as italicizing) is a matter of setting some Level 0 tags. The shearing tags, OT_ShearSin and OT_ShearCos, specify the shearing angle, or the angle at which the typeface is italicized. This angle refers to the angle that results from rotating the vertical axis clockwise. The angle can range from -45 to 45 degrees (inclusive). Like the rotation angle, the shearing angle is represented as a sine and cosine value that must correspond to the same angle and must fall into normal bounds for sine and cosine values. Also like the rotation angle sine and cosine tags, an application must set both the OT_ShearSin and OT_ShearCos tags, in that order. By default, the shearing value is zero degrees meaning there is no shearing (OT_ShearSin = 0x00000000, OT_ShearCos = 0x00010000).

Other Level 0 Tags

There are several other Level 0 tags:

OT_DotSize. This tag specifies the X and Y size of the target device's dots. The X and Y DPI imply a dot size. For example, at 300 X and 300 Y DPI, the resolution implied dot size is 1/300 inches by 1/300 inches. For some devices (like some dot matrix printers), the size of the output dot does not match its resolution implied size. To a degree, the IntelliFont engine can account for this. The dot size is represented as a percentage of the dot's resolution implied size. The X percentage is in the tag value's upper word, and the Y percentage is in the tag value's lower word.

OT_SetFactor. This tag distorts the width of a typeface by changing the width of the em square. The scaling engine changes the em width to this tag's value. The value is a fixed point binary fraction.

OT_EmboldenX/OT_EmboldenY. These tags specify the algorithmic emboldening factor in the X and Y direction, respectively. The tag values are fixed point two's complement binary numbers. The units are measured in ems. Emboldening values above zero embolden the typeface. Emboldening values below zero lighten the typeface. By default, both values are zero.

OT_GlyphWidth. This tag's value specifies a width for the current typeface. It is a fraction of an em represented as a fixed point binary number. If this value is set to something besides 0.0, all glyphs will have this width. To turn off the constant width, set `OT_GlyphMap` back to 0.0 (its default value).

The Otag File Tags

The Outline Tag (otag) file contains a number of tags that describe a font outline to the Diskfont library. The purpose of most of these tags is to allow Diskfont to attribute styles to a typeface when loading a font outline as a standard Amiga system font. Most applications that use the scaling engine will not need to worry about the meaning of the majority of these tags, but they are described below. The following tags are Level 1 tags and must be present in every otag file:

OT_FileIdent. Every valid otag file starts with this tag. Its value is the size of the file. It doesn't really have anything to do with the definition of the typeface, but it does serve as a way to check the validity of the otag file.

OT_Engine. This tag's value points to a string naming the font scaling engine. For example, the OT_Engine tag in *fonts:CGTimes.otag* is "bullet".

OT_Family. This tag's value points to a string naming the typefaces font family. For example, the OT_Family in *fonts:CGTimes.otag* is "bullet".

OT_SymbolSet. This tag's value is a two byte ASCII code for this typeface's symbol set. This tells the system which symbol set to use to map the Amiga character set to the Compugraphic character set. The symbol set for most CG fonts designed for use with the Amiga is "L1", which stands for Latin1. The exception is the CG fonts from Gold Disk, Inc. They use the "GD" (Gold Disk) symbol set.

OT_YSizeFactor. For traditional Amiga fonts, the font size does not include any spacing on top or bottom of the typeface--the Amiga doesn't consider it part of the font. CG fonts include spacing on the top and bottom of their typefaces. This tag's value is a ratio of the point height of a typeface to its "black" height (the point height minus the space on the typeface's top and bottom). The high word is the point height factor and the low word is the black height factor. Note that these values form a ratio. Individually, they do not necessarily reflect any useful value.

OT_SerifFlag. If this tag's value is TRUE, this typeface has serifs.

OT_StemWeight. This tag's value can be anywhere from 0 through 255 and indicates a nominal weight or "boldness" to the typeface. The *<diskfont/diskfonttag.h>* include file defines a set of ratings for this tag's value. See that file for more details. When the Diskfont library opens an outline font, it uses this value to determine if a typeface is bold.

OT_SlantStyle. The *<diskfont/diskfonttag.h>* include file defines a set of three possible values for this tag's value. See that file for more details. When the Diskfont library opens an outline font, it uses this value to determine if a typeface is italicized/obliqued.

OT_HorizStyle. This tag's value can be anywhere from 0 through 255 and indicates a nominal width rating to the typeface. The *<diskfont/diskfonttag.h>* include file defines a set of ratings for this tag's value. See that file for more details. When the Diskfont library opens an outline font, it uses this value to determine if a typeface is extended.

OT_AvailSizes. This tag's value points to a sorted list of UWORDs. The first UWORD is the number of entries in the sorted list. The remaining UWORDs are the font sizes that the Diskfont library lists when calling AvailFonts().

OT_SpecCount. This tag's value is a number of spec tags that follow it. A spec tag is private to the scaling engine.

The following are Level 2 tags. They may also be in an otag file but are not required:

OT_BName. This tag points to a string naming the bold variant of this typeface. For example, the *fonts:CGTimes.otag* file lists "CGTimesBold" as its bold variant.

OT_IName. This tag points to a string naming the italic variant of this typeface.

OT_BIName. This tag points to a string naming the bold italic variant of this typeface.

OT_SpaceWidth. This tag's value is the width of the space character at 250 points (where there are 72.307 points in an inch). The width is in Design Window Units (DWUs). One DWU is equal to 1/2540 inches. To convert to X pixels:

```
OT_SpaceWidth   pointsize
------------- * --------- *  XDPI = spacewidth in pixels (X dots)
    2540          250
```

OT_IsFixed. If this tag's value is TRUE, every glyph in this typeface has the same character advance (a fixed width).

OT_InhibitAlgoStyle. This tag's value is a bitmask that is compatible with the ta_Style field of the TextAttr structure (defined in *<graphics/text.h>*). This tag tells which styles cannot be added to a typeface algorithmically. For example, if the FSF_BOLD bit in OT_InhibitAlgoStyle is set and a user asks for a bold version of the typeface, the *diskfont.library* (or an application) can add that style algorithmically.

At present there are no Level 3 tags.

About the Examples

This article contains two code examples. The first, *Rotate*, rotates a user-specified glyph around a central point. By default, it rotates a 36 point 'A' using the font *fonts:CGTimes.font*. If *Rotate* finds an AmigaDOS environment variable called "XYDPI", it will use the X and Y DPI it finds in that variable as the default target device DPI (see the description of the Level 0 OT_DeviceDPI tag). If that variable is not defined, Rotate will use an XDPI of 68 and a YDPI of 27 which, nominally, is the X and Y DPI of a standard Hires display.

The second example, *View*, displays a file using the same defaults as *Rotate*. *View* utilizes kerning pairs to display its glyphs. Because *View* only considers "visible" characters, it ignores characters that have widths but no glyph. The result is, *View* doesn't print any space characters. If *View* were smarter, it would ask the scaling engine for a width list so it could properly advance the current pointer when it comes across a space or some other character without a glyph.

Notice that *View* uses a slightly modified version of *BulletMain.c* called *BulletMainFile.c*. Only *BulletMain.c* appears in the example code. The only significant difference between the two is that *BulletMainFile.c* obtains a file name for *View* to display.

```
;/* Engine.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 Engine.c
quit ;*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dostags.h>
#include <dos/dos.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <utility/tagitem.h>
#include <string.h>

#include <clib/dos_protos.h>
#include <clib/exec_protos.h>
#include <clib/utility_protos.h>
#include <clib/bullet_protos.h>

#define OTAG_ID 0x0f03    /* this really belongs in <diskfont/diskfont.h>, */
                          /* but it's not there, yet.                      */

extern UBYTE   *librarystring; /* *.library*, defined in BulletMain.c. */

struct TagItem *AllocOtag(STRPTR);
void            FreeOtag(void *);
struct Library *OpenScalingLibrary(struct TagItem *);
void            CloseScalingLibrary(struct Library *);
struct GlyphEngine *GetGlyphEngine(struct TagItem *, STRPTR);
void            ReleaseGlyphEngine(struct GlyphEngine *);

#define BUFSIZE     256

extern struct Library *BulletBase, *UtilityBase;

/*****************************************************************************/
/* open the otag file, allocate a buffer, read the file into the buffer, verify that */
/* the file is OK, relocate all of the address relocation tags, close the otag file. */
/*****************************************************************************/
struct TagItem *
AllocOtag(STRPTR otagname)
{
  BPTR            otfile;
  struct TagItem *ti, *tip, *returnti;
  struct FileInfoBlock *fib;

  ti = NULL;

  if (fib = AllocDosObject(DOS_FIB, NULL))    /* The FileInfoBlock of the OTAG file */
  {                                           /* contains the file's size.          */
    if (otfile = Open(otagname, MODE_OLDFILE))
    {
      if (ExamineFH(otfile, fib))
      {
        if (returnti = (struct TagItem *) AllocVec(fib->fib_Size, MEMF_CLEAR))
        {
          if (Read(otfile, (UBYTE *) returnti, fib->fib_Size))
          {
            if ((returnti->ti_Tag == OT_FileIdent)                /* Test to see if */
                && (returnti->ti_Data == (ULONG) fib->fib_Size)) /* the OTAG file  */
            {                                                     /* is valid.      */
              tip = returnti;
              while (ti = NextTagItem(&tip))     /* Step through and relocate tags */
              {
                if (ti->ti_Tag & OT_Indirect)
                {
                  ti->ti_Data = (ULONG) returnti + ti->ti_Data;
                }
              }
            }
          }
        }
      }
      Close(otfile);
    }
    FreeDosObject(DOS_FIB, fib);
  }
  return (returnti);
}

/*****************************************************************************/
/*********** Deallocates resources allocated by AllocOtag(). *************/
/*****************************************************************************/
void
FreeOtag(void *vector)
{
  FreeVec(vector);
}


/*****************************************************************************/
/******* Scans through a TagList looking for an scaling engine name. *********/
/*******            If it finds one, it opens that library.          *********/
/*****************************************************************************/
struct Library *
OpenScalingLibrary(struct TagItem * ti)
{
  STRPTR          enginename;
  UBYTE           libnamebuffer[BUFSIZE];

  if (enginename = (STRPTR) GetTagData(OT_Engine, NULL, ti))
  {
    strcpy(libnamebuffer, enginename);
    strcat(libnamebuffer, librarystring);

    return (OpenLibrary(libnamebuffer, 0L));
  }
}

/*****************************************************************************/
/******* Deallocates resources allocated by OpenScalingLibrary(). *********/
/*****************************************************************************/
void
CloseScalingLibrary(struct Library * bbase)
{
  CloseLibrary(bbase);
}


/*****************************************************************************/
/* Open the glyph engine, give it the tags from the otag file, and set up */
/* a default device dpi so it doesn't crash if someone forgets to set it. */
/*****************************************************************************/
struct GlyphEngine *
GetGlyphEngine(struct TagItem * ti, STRPTR otagpath)
{
  struct GlyphEngine *ge = NULL;
  BOOL            ok = TRUE;

  if (ge = OpenEngine())
  {
    ok = FALSE;
    if (SetInfo(ge,
                OT_OTagList, ti,
                OT_OTagPath, otagpath,
                TAG_END) == OTERR_Success)
    {
      if (SetInfo(ge,
                  OT_DeviceDPI, ((ULONG) 77) << 16 | 75,
                  TAG_END) == OTERR_Success)
      {
        ok = TRUE;
      }
    }
  }

  if (!ok)
  {
    CloseEngine(ge);
    ge = NULL;
  }
  return (ge);
}
```

```
/*****************************************************************************/
/********* Deallocates resources allocated by GetGlyphEngine(). **************/
/*****************************************************************************/
void
ReleaseGlyphEngine(struct GlyphEngine * ge)
{
  CloseEngine(ge);
}
```

```
;/* BulletMain.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 BulletMain.c
quit ;*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/rdargs.h>
#include <dos/dos.h>
#include <dos/var.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <utility/tagitem.h>
#include <string.h>
#include <graphics/displayinfo.h>
#include <intuition/intuition.h>
#include <intuition/screens.h>

#include <clib/dos_protos.h>
#include <clib/graphics_protos.h>
#include <clib/exec_protos.h>
#include <clib/utility_protos.h>
#include <clib/bullet_protos.h>
#include <clib/intuition_protos.h>

#define OTAG_ID 0x0f03

#ifdef LATTICE
int CXBRK(void) { return (0);} /* Disable Lattice CTRL/C handling */
int chkabort(void) { return (0);}
#endif

UBYTE   *readargsstring = "FontName,Size/N,XDPI/N,YDPI/N,CharCode/N,CharCode2/N\n";
UBYTE   *librarystring = ".library";
UBYTE   *fontstring     = "fonts:cgtimes.font";
UBYTE   *dpivarname = "XYDPI";  /* Name of an X/Y DPI environment variable. */
                                /* If this ENV: variable exists, this code  */
                                /* will use the X and Y DPI stored there.   */
                                /* This code will also save the X and Y DPI */
                                /* in XYDPI if the user supplies a DPI.     */
                                /* XYDPI encodes the DPI just like the      */
                                /* OT_DeviceDPI tag.                        */
extern struct TagItem *AllocOtag(STRPTR);
extern void     FreeOtag(void *);
extern struct Library *OpenScalingLibrary(struct TagItem *);
extern void     CloseScalingLibrary(struct Library *);
extern struct GlyphEngine *GetGlyphEngine(struct TagItem *, STRPTR);
extern void     ReleaseGlyphEngine(struct GlyphEngine *);
extern void
BulletExample(struct GlyphEngine *,
              struct Window *,
              struct RastPort *, ULONG, ULONG, ULONG, ULONG, ULONG);

#define BUFSIZE     256

#define NUM_ARGS    6    /* Arguments for ReadArgs(). */
#define FONT_NAME   0
#define SIZE        1
#define XDPI        2
#define YDPI        3
#define CODE        4
#define CODE2       5
LONG            args[NUM_ARGS];
struct RDargs  *myrda;

struct Library *BulletBase, *UtilityBase, *GfxBase, *IntuitionBase;

UBYTE           buf[BUFSIZE];
BPTR            fontfile, dpifile;
UBYTE          *otagname;
UWORD           fchid;

struct GlyphEngine *ge;

struct DrawInfo *drawinfo;
struct RastPort rp;
```

```
void            main(int argc, char **argv)
{
  struct TagItem *ti;
  struct GlyphEngine *ge;
  struct Window  *w;

  UBYTE           xydpi[5];

  ULONG           defpointheight = 36;   /* Default values for ReadArgs() */
  ULONG           defxdpi = 68;          /* variables.                    */
  ULONG           defydpi = 27;
  ULONG           defcode = (ULONG) 'A';
  ULONG           defcode2 = 0;

  if (GfxBase = OpenLibrary("graphics.library", 37L))
  {
    if (IntuitionBase = OpenLibrary("intuition.library", 37L))
    {
      if (myrda = ReadArgs(readargsstring, args, NULL))
      {
        if (args[XDPI] && args[YDPI]) /* If the user sets the DPI from the command  */
        {             /* line, make sure the environment variable also gets changed. */
          *(ULONG *)xydpi = ( (*(LONG *) args[XDPI]) << 16 | (*(ULONG *) args[YDPI]) );
          SetVar(dpivarname, xydpi, 5,
              GVF_GLOBAL_ONLY | GVF_BINARY_VAR | GVF_DONT_NULL_TERM);
        }
        else                          /* If the user did NOT set the X OR Y DPI...*/
        {
          args[XDPI] = (LONG) &defxdpi; /* ...set to default values and look for    */
          args[YDPI] = (LONG) &defydpi; /* an environment variable called 'XYDPI'.  */
                                        /* Read the environment variable, */
          if (GetVar(dpivarname, xydpi, 5,        /* XYDPI, if it exists.          */
              GVF_GLOBAL_ONLY | GVF_BINARY_VAR | GVF_DONT_NULL_TERM))
          {
            if ( (*(ULONG *)xydpi & 0xFFFF0000) && (*(ULONG *)xydpi & 0x0000FFFF) )
            {     /* Make sure the environment variable is OK to use by making sure */
                  /* that neither X or YDPI is zero. If XYDPI is OK, use it as the  */
              defxdpi = ((*(ULONG *)xydpi) & 0xFFFF0000) >> 16;       /* default.  */
              defydpi = (*(ULONG *)xydpi) & 0x0000FFFF;
            }
          }
        }
        if (!(args[SIZE]))
          args[SIZE] = (LONG) &defpointheight;
        if (!(args[CODE]))
          args[CODE] = (LONG) &defcode;
        if (!(args[CODE2]))
          args[CODE2] = (LONG) &defcode2;
        if (!(args[FONT_NAME]))
          args[FONT_NAME] = (LONG) fontstring;
                                       /* Open the '.font' file which contains  */
                                       /* the FontContentsHeader for this font. */
        if (fontfile = Open((STRPTR) args[FONT_NAME], MODE_OLDFILE))
        {
          if (Read(fontfile, &fchid, sizeof(UWORD)))
          {
            if (fchid == OTAG_ID)             /* Does this font have an .otag file? */
            {
              strcpy(buf, (STRPTR) args[FONT_NAME]); /* Put together the otag file  */
              if (otagname = &(buf[strlen(buf) - 4]))/* name from the .font file.   */
              {
                strcpy(otagname, "otag");
                if (UtilityBase = OpenLibrary("utility.library", 37L))
                {
                  if (ti = AllocOtag(buf))      /* open the otag file and copy its  */
                  {                             /* tags into memory.                */
                    if (BulletBase = OpenScalingLibrary(ti)) /* Pass the function   */
                    {                                        /* the OTAG tag list which it*/
                                                             /* needs to open the scaling */
                      if (ge = GetGlyphEngine(ti, buf))/* library.  Open the        */
                      {                                /* library's scaling engine. */
                        if (w = OpenWindowTags(NULL,
                                               WA_Width,         640,
                                               WA_Height,        200,
                                               WA_SmartRefresh,  TRUE,
                                               WA_SizeGadget,    FALSE,
                                               WA_CloseGadget,   TRUE,
                                               WA_IDCMP,         NULL,
                                               WA_DragBar,       TRUE,
                                               WA_DepthGadget,   TRUE,
                                               WA_Title,         (ULONG)argv[0],
                                               TAG_END))
                        {
                          rp = *(w->RPort); /* Clone window's RastPort.  The second */
                                            /* Rastport is for rendering with the   */
                                            /* background color.                    */

                          if (drawinfo = GetScreenDrawInfo(w->WScreen)) /* Get the  */
                          {               /* Screen's DrawInfo to get its pen colors. */
                            SetAPen(w->RPort, drawinfo->dri_Pens[TEXTPEN]);
                            SetAPen(&rp, drawinfo->dri_Pens[BACKGROUNDPEN]);
                            FreeScreenDrawInfo(w->WScreen, drawinfo);

                            BulletExample(ge, w, &rp, *(ULONG *) args[SIZE],
                                          *(ULONG *) args[XDPI],
                                          *(ULONG *) args[YDPI],
                                          *(ULONG *) args[CODE],
                                          *(ULONG *) args[CODE2]);
                          }
                          CloseWindow(w);
                        }
                        ReleaseGlyphEngine(ge);
                      }
                      CloseScalingLibrary(BulletBase);
                    }
                    FreeOtag(ti);
                  }
                  CloseLibrary(UtilityBase);
                }
              }
            }
          }
          Close(fontfile);
        }
        FreeArgs(myrda);
      }
      CloseLibrary(IntuitionBase);
    }
    CloseLibrary(GfxBase);
  }
}
```

```
;/* Rotate.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 Rotate.c
Blink FROM LIB:c.o,BulletMain.o,engine.o,Rotate.o TO Rotate LIBRARY
LIB:LC.lib,LIB:Amiga.lib
quit ;*/

#include <exec/types.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <graphics/gfx.h>
#include <graphics/regions.h>
#include <utility/tagitem.h>
#include <intuition/intuition.h>
#include <devices/timer.h>

#include <clib/alib_stdio_protos.h>
#include <clib/alib_protos.h>
#include <clib/bullet_protos.h>
#include <clib/exec_protos.h>
#include <clib/layers_protos.h>
#include <clib/graphics_protos.h>
#include <clib/intuition_protos.h>

extern struct Library *BulletBase, *UtilityBase, *GfxBase, *IntuitionBase;
struct Library *LayersBase;
void            BulletExample(struct GlyphEngine *,
                              struct Window *,
                              struct RastPort *,
                              ULONG, ULONG, ULONG, ULONG, ULONG);

UBYTE          *vers = "\e0$VER: Rotate 38.9";

#define TABLE_ENTRIES 24
#define SINE_INDEX    0
#define COSINE_INDEX  1

/* precalculated sine and cosine */
LONG            table[TABLE_ENTRIES][2] =
{
  (0x0, 0x10000),             /* 0   degrees */ /* Notice that the sine and cosine  */
  (0x424e, 0xf747),           /* 15  degrees */ /* values have to correspond to the */
  (0x8000, 0xddb4),           /* 30  degrees */ /* same angle.  The IntelliFont     */
  (0xb505, 0xb505),           /* 45  degrees */ /* engine will have severe mental   */
  (0xddb4, 0x8000),           /* 60  degrees */ /* problems if the values aren't    */
  (0xf747, 0x424e),           /* 75  degrees */ /* close to representing the same   */
  (0x10000, 0x0),             /* 90  degrees */ /* angle.                           */
  (0xf747, 0xffffbdbe),       /* 105 degrees */
  (0xddb4, 0xffff8000),       /* 120 degrees */
  (0xb505, 0xffff4afb),       /* 135 degrees */
  (0x8000, 0xffff224c),       /* 150 degrees */
  (0x424e, 0xffff08b9),       /* 165 degrees */
  (0x0, 0xffff0000),          /* 180 degrees */
  (0xffffbdbe, 0xffff08b9),   /* 195 degrees */
  (0xffff8000, 0xffff224c),   /* 210 degrees */
  (0xffff4afb, 0xffff4afb),   /* 225 degrees */
  (0xffff224c, 0xffff8000),   /* 240 degrees */
  (0xffff08b9, 0xffffbdbe),   /* 255 degrees */
  (0xffff0000, 0x0),          /* 270 degrees */
  (0xffff08b9, 0x424e),       /* 285 degrees */
  (0xffff224c, 0x8000),       /* 300 degrees */
  (0xffff4afb, 0xb505),       /* 315 degrees */
  (0xffff8000, 0xddb4),       /* 330 degrees */
  (0xffffbdbe, 0xf747)        /* 345 degrees */
};

struct Rectangle rectangle;
struct Region    *region;

void
BulletExample(struct GlyphEngine * ge,
              struct Window * w, struct RastPort * rp,
     ULONG pointheight, ULONG xdpi, ULONG ydpi, ULONG unicode, ULONG unicode2)
{
  struct GlyphMap *gm;
  PLANEPTR         tempbitmap;
  ULONG            centerx, centery, x, y, dx, dy, sin, cos, oldx, oldy, olddx,
                   olddy, emheight, emwidth;
  ULONG            i = 0;

  struct IntuiMessage *mymsg;
  BOOL             done = FALSE;

  if (pointheight > 180) pointheight = 180;

  if (SetInfo(ge,
              OT_DeviceDPI, ((ULONG) xdpi) << 16 | ydpi,
              TAG_END) != OTERR_Success)
        return;

  emheight = (pointheight * ydpi) / 72; /* Calculate the pixel dimensions */
  emwidth  = (pointheight * xdpi) / 72; /* of the EM square.              */
  centerx  = w->BorderLeft + emheight;
  centery  = w->BorderTop  + emwidth;

  dx = (2 * emwidth) + w->BorderLeft + w->BorderRight;  /* Calculate window size to */
  dy = (2 * emheight) + w->BorderTop + w->BorderBottom; /* fit around glyph com-    */
  dx = (dx > 640) ? 640 : dx;                           /* fortablely.              */
  dy = (dy > 200) ? 200 : dy;
  dx = (dx < 80) ? 80: dx;
  dy = (dy < 50) ? 50: dy;

  if (ModifyIDCMP(w, IDCMP_CHANGEWINDOW))
  {
    ChangeWindowBox(w, w->LeftEdge, w->TopEdge, dx, dy);        /* Set window size  */
    WaitPort(w->UserPort);                                      /* and wait for the */
    while (mymsg = (struct IntuiMessage *) GetMsg(w->UserPort)) /* dimension change */
      ReplyMsg((struct Message *) mymsg);                       /* to take place.   */
    if (!(ModifyIDCMP(w, NULL))) return;  /* Quit if there is a problem with IDCMP. */
  }

  x = centerx; /* calculate original rendering position. */
  y = centery;
  dx = 1;   /* Since dx and dy are no longer necessary for figuring out the window  */
  dy = 1;   /* dimensions, I use them to measure the full width and height of the   */
            /* glyph bitmap supplied by bullet.  I need this to erase the glyph.    */

  if (LayersBase = OpenLibrary("layers.library", 37L)) /* These lines are       */
  {                                                    /* here to install       */
    rectangle.MinX = w->BorderLeft;                    /* a clipping            */
    rectangle.MinY = w->BorderTop;                     /* region to the         */
    rectangle.MaxX = w->Width  - w->BorderRight - 1;   /* window to keep        */
    rectangle.MaxY = w->Height - w->BorderBottom - 1;  /* the glyph within      */
                                                       /* window bounds.        */
    if (region = NewRegion())                          /* For more information, */
    {                                                  /* see the "Layers"      */
      if (OrRectRegion(region, &rectangle))            /* chapter of the        */
      {                                                /* RKRM: Libraries       */
        InstallClipRegion(w->WLayer, region);          /* Manual.               */

        if (SetInfo(ge,
                    OT_GlyphCode, unicode,                        /* Set the glyph to */
                    OT_PointHeight, (ULONG) pointheight << 16,    /* rotate and its   */
                    TAG_END) == OTERR_Success)                    /* pointsize.       */
        {
          SetDrMd(w->RPort, JAM1);
          if (tempbitmap = AllocRaster(640, 200))
          {
            if (ModifyIDCMP(w, IDCMP_CLOSEWINDOW)) /* Turn on close window reports  */
            {                                      /* so the example can quit.      */
              for (i = 0; done == FALSE; i++)
              {
                if (i == TABLE_ENTRIES)
                  i = 0;

                sin = table[i][SINE_INDEX];   /* Step through the sine/cosine array */
                cos = table[i][COSINE_INDEX]; /* 360 degrees @ 15 degree increments */

                if (SetInfo(ge,                       /* Set the current rotation angle. */
                            OT_RotateSin, sin,
                            OT_RotateCos, cos,
                            TAG_END) == OTERR_Success)
                {
```

```
                if ((ObtainInfo(ge, OT_GlyphMap, &gm, TAG_END)) == OTERR_Success)
                {
                  oldx = x;                 /* Calculate the dimension and position */
                  oldy = y;                 /* of the new glyph's bitmap and save   */
                  olddx = dx;               /* the old values so we can erase the   */
                  olddy = dy;               /* glyph that is still on the screen.   */
                  x = centerx - gm->glm_X0;
                  y = centery - gm->glm_Y0;
                  dx = gm->glm_BMModulo * 8;
                  dy = gm->glm_BMRows;

                  CopyMem(gm->glm_BitMap,/* Copy the glyph's bitmap into Chip RAM */
                          tempbitmap,    /* so we can blit it into a RastPort.    */
                          gm->glm_BMModulo * gm->glm_BMRows);

                                                      /* Erase the old glyph. */
                  RectFill(rp, oldx, oldy, oldx + olddx, oldy + olddy);

                  WaitBlit();                 /* Wait for the old glyph to erase. */

                  BltTemplate(                   /* Blit the new glyph into the   */
                              tempbitmap,        /* window's RastPort.            */
                              0,
                              gm->glm_BMModulo,
                              w->RPort,
                              x,
                              y,
                              dx,
                              dy);           /* Running several instances of this */
                                             /* example simultaneously can really */
                                             /* slow the system, so we give other */
                  TimeDelay(UNIT_VBLANK, 0, 250000);  /*   tasks a chance to run. */
                  ReleaseInfo(ge, OT_GlyphMap, gm, TAG_END);
                }
              }                              /* Check for a CLOSEWINDOW message. */
              while (mymsg = (struct IntuiMessage *) GetMsg(w->UserPort))
              {
                ReplyMsg((struct Message *) mymsg);
                done = TRUE;
              }
            }
          }
          FreeRaster(tempbitmap, 640, 200);
        }
      }
    }
    DisposeRegion(region);
  }
  CloseLibrary(LayersBase);
 }
}
```

```
;/* View.c - Execute me to compile me with SAS/C 5.10a
LC -cfistq -v -y -j73 View.c
Blink FROM LIB:c.o,BulletMainFile.o,engine.o,View.o TO View LIBRARY
LIB:LC.lib,LIB:Amiga.lib
quit ;*/


#include <exec/types.h>
#include <exec/memory.h>
#include <diskfont/diskfonttag.h>
#include <diskfont/diskfont.h>
#include <diskfont/glyph.h>
#include <diskfont/oterrors.h>
#include <graphics/gfx.h>
#include <utility/tagitem.h>
#include <intuition/intuition.h>
#include <devices/timer.h>
#include <dos/dos.h>

#include <clib/alib_stdio_protos.h>
#include <clib/alib_protos.h>
#include <clib/bullet_protos.h>
#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/graphics_protos.h>
#include <clib/intuition_protos.h>

UBYTE          *vers = "\e0$VER: View 38.6";

extern struct Library *BulletBase, *UtilityBase, *GfxBase, *IntuitionBase;
void            BulletExample(struct GlyphEngine *,
                              struct Window *,
                              struct RastPort *,
                              ULONG, ULONG, ULONG, STRPTR);
struct GlyphMap *gm;
PLANEPTR        tempbitmap;
struct IntuiMessage *mymsg;
UBYTE          *viewfilebuf, *currchar;
ULONG           currposition, emheight, emwidth, x, y;

BPTR            viewfile;
LONG            actual;
struct Task    *mytask;
struct FileInfoBlock *fib;

void
BulletExample(struct GlyphEngine * ge,
              struct Window * w, struct RastPort * rp,
              ULONG pointheight, ULONG xdpi, ULONG ydpi, STRPTR viewfilename)
{
  UWORD           wlimitx, wlimity, newwidth;
  FIXED           kern;

  wlimitx = w->Width  - w->BorderRight  - 2; /* The X and Y extent of the window */
  wlimity = w->Height - w->BorderBottom - 2; /* that we can draw into.           */

  if (SetInfo(ge,                                   /* Set up the X and Y DPI of */
              OT_DeviceDPI, ((ULONG)xdpi) << 16 | ydpi,/* target raster.  Neither */
              OT_PointHeight, (ULONG)pointheight << 16,/* of these can be zero!   */
              TAG_END) != OTERR_Success)            /* BulletMainFile.c checks   */
    return;                                         /* for zero.                 */

  if (viewfile = Open(viewfilename, MODE_OLDFILE))/* Open the ASCII file to display.*/
  {
    if (fib = AllocDosObject(DOS_FIB, NULL))      /* Find out how big the display   */
    {                                             /* file is by looking at its      */
      if (ExamineFH(viewfile, fib))               /* FileInfoBlock.  Allocate that  */
      {                                           /* Much memory.                   */
        if (viewfilebuf = (UBYTE *) AllocVec(fib->fib_Size, MEMF_CLEAR))
        {
          if (Read(viewfile, (UBYTE *) viewfilebuf, fib->fib_Size))    /* Read the  */
          {                                       /* whole file into its buffer. */
            SetDrMd(w->RPort, JAM1);
            if (tempbitmap = AllocRaster(640, 200)) /* Allocate some Chip RAM space */
            {       /* where we can temporarily store the glyph so we can blit it.  */
              if (ModifyIDCMP(w, IDCMP_CLOSEWINDOW))  /* Turn on the Close gadget.  */
              {
```

```
emheight = (pointheight * ydpi) / 72; /* Calculate the dimensions   */
emwidth = (pointheight * xdpi) / 72;  /* of the EM square in screen */
     /* pixels. This is necessary because bullet.library measures   */
     /* character widths and kerning values as fractions of an EM.  */
     /* An EM (pronounced like the letter 'M') is a measure of      */
     /* distance that is equal to the point size of a typeface      */
     /* (which means one EM is not constant across different type   */
     /* sizes).  For a 72 point typeface, one EM = 72 points which  */
     /* approximately equals one inch.                              */

x = w->BorderLeft + 2;           /* Calculate the starting point    */
y = w->BorderTop + 2 + emheight; /* for glyph rendering.            */

                        /* Step through each character in the buffer. */
for (currposition = 0; currposition < fib->fib_Size; currposition++)
{
               /* Set the current glyph, which is the one we'll be */
               /* rendering in this interation of the loop, and    */
               /* the secondary glyph, which, besides being the    */
               /* next glyph we will render, is necessary to find  */
               /* the proper kerning value between the glyphs.     */
               /* Notice that this example does not account for    */
               /* the presence of non-printables (carriage return, */
               /* DEL, etc.) which effects the kerning.  A real    */
  if (SetInfo(ge,  /* application should consider these.           */
              OT_GlyphCode,  (ULONG) viewfilebuf[currposition],
              OT_GlyphCode2, (ULONG) viewfilebuf[currposition + 1],
              TAG_END) == OTERR_Success)
  {
    kern = 0;          /* Find the kerning adjustment between glyph1 */
                       /* and glyph2.  This example doesn't account  */
                       /* for the validity of the glyphs.            */
    ObtainInfo(ge, OT_TextKernPair, &kern, TAG_END);

                                  /* Ask the scaling engine for the */
                                  /* bitmap for the current glyph.  */
    if ((ObtainInfo(ge, OT_GlyphMap, &gm, TAG_END)) == OTERR_Success)
    {
       /* Calculate the width of the current character including    */
       /* any kerning adjustment.  Because the width is represented */
       /* as a fixed point binary fraction of an EM, this needs to  */
       /* be converted to a width in screen pixels.                 */
      newwidth = ((gm->glm_Width - kern) * emwidth) / 65536;


      if ((x + newwidth) > wlimitx)  /* Make sure the glyph gets    */
      {                              /* rendered inside the window  */
                                     /* bounds.                     */
        x = w->BorderLeft + 2;
        y += emheight;
        if (y > wlimity)   /* If the text goes beyond the bottom of */
        {                  /* the window, clear the window and move */
                           /* the current rendering position to the */
                           /* upper left.                           */
          y = w->BorderTop + 2 + emheight;
          RectFill(rp, w->BorderLeft, w->BorderTop, wlimitx, wlimity);
        }
      }

      CopyMem(gm->glm_BitMap,/* Copy the raw bitmap to chip memory. */
              tempbitmap,
              gm->glm_BMModulo * gm->glm_BMRows);

      BltTemplate(           /* Render the glyph using the blitter  */
                             /* and the RastPort settings.          */
                  (PLANEPTR) (((ULONG) tempbitmap)
                    + (gm->glm_BMModulo * gm->glm_BlackTop)
                      + ((gm->glm_BlackLeft >> 4) << 1)),
                  gm->glm_BlackLeft & 0xF,
                  gm->glm_BMModulo,
                  w->RPort,
                  x - gm->glm_X0 + gm->glm_BlackLeft,
                  y - gm->glm_Y0 + gm->glm_BlackTop,
                  gm->glm_BlackWidth,     /* glm_X0 & Y0 are used  */
                  gm->glm_BlackHeight);   /* to make the example a */
                   /* little simpler.  They are not as accurate as */
                   /* using glm_XOrigin and glm_YOrigin in con-    */
                                  /* juntion with fractional width and kerning    */
                                  /* values.                                      */

                    x += newwidth;
                    ReleaseInfo(ge, OT_GlyphMap, gm, TAG_END);
                  }
                  while (mymsg = (struct IntuiMessage *) GetMsg(w->UserPort))
                  {                                        /* Did the user hit the */
                    ReplyMsg((struct Message *) mymsg);    /* Close Gadget?        */
                    currposition = fib->fib_Size + 1;
                  }
                }
              }
            }
            FreeRaster(tempbitmap, 640, 200);
          }
        }
        FreeVec(viewfilebuf);
      }
    }
    FreeDosObject(DOS_FIB, fib);
  }
  Close(viewfile);
 }
}
```

Memory Limits for Various CDTV Configurations

by Adam Levin-Delson

RAM space is at a premium in the CDTV, so for the developer who is creating CDTV titles, it is important to know what the limits of system memory are. This is especially true for developers who want their titles to work on a variety of CDTV/A500 configurations. To help assist developers in making titles work on the maximum number of configurations, the following chart shows a variety of hardware configurations capable of playing CDTV discs.

For comparison purposes, all machines were allowed to boot far enough to open the Workbench screen (in the default size) and a CLI/Shell window. This represents a reasonable starting point for developers to estimate if their application will fit into a CDTV's memory (To properly ascertain the amount of free memory in a particular CDTV/A500(+) at run time, use the Exec library's AvailMem() function). These results are not guaranteed for any particular configuration, they can and will change depending upon many variables, including additional hardware, overscan or interlaced screens, etc. Use this information only as a guideline. Do not assume that these result will remain constant in future releases of the OS.

WARNING: A title that consumes too much memory on a current configuration may not work in the future if a new release of the OS consumes even a little more memory.

Important note about Workbench V1.3 and V 2.0. Under 1.3, the Workbench screen and initial CLI always open upon booting with an AmigaDOS disc. Under 2.0, the system only opens the Workbench screen when necessary, and only opens an initial CLI if the *Startup-sequence* attempts to output text. The *C:Avail* program first calculates the amount of free RAM and then prints out the results. Under 2.0, this means the amount of memory printed by *Avail* would differ from the amount of memory actually free by the amount needed to create the Workbench screen and initial CLI.

Machine	NTSC/PAL	1.3/2.0	Peripherals	*	Available	In-Use	Maximum	Largest
CDTV	NTSC	1.3	internal HD floppy	c	799448	240704	1040152	797312
CDTV	NTSC	1.3	floppy	c	886688	153464	1040152	710472
CDTV	NTSC	2.0	floppy	c	840616	206936	1047552	689200
CDTV	NTSC	1.3	SCSI card	c	881248	158904	1040152	704904
CDTV	NTSC	1.3	floppy	c	878824	161328	1040152	878624
CDTV	PAL	2.0	floppy	c	807528	239992	1047520	792880
CDTV	PAL	2.0	none	c	828784	218768	1047552	824736
CDTV	PAL	1.3	floppy	c	877936	162216	1040152	710552
CDTV	PAL	1.3	internal HD floppy	c	790488	249664	1040152	788352
A500/570	NTSC	1.3	internal FD	c	466256	56976	523232	466208
				f	412728	104160	516888	412552
				t	878984	161136	1040120	466208
A500/570	PAL	1.3	internal FD	c	457296	65936	523232	457248
				f	412704	104184	516888	412064
				t	870000	170120	1040120	457248
A500+/570	PAL	2.0	internal FD	c	809744	237776	1047520	795144
A500+	PAL	2.0	internal FD	c	839688	207864	1047552	838648
CDTV	NTSC	1.3	floppy	c	886688	153464	1040152	710472
CDTV	NTSC	2.0	floppy	c	840616	206936	1047552	689200
CDTV	NTSC	1.3	SCSI card	c	881248	158904	1040152	704904
CDTV	NTSC	1.3	floppy	c	878824	161328	1040152	878624
CDTV	PAL	2.0	floppy	c	807528	239992	1047520	792880
CDTV	PAL	2.0	none	c	828784	218768	1047552	824736
CDTV	PAL	1.3	floppy	c	877936	162216	1040152	710552
A500/570	NTSC	1.3	internal floppy	c	466256	56976	523232	466208
				f	412728	104160	516888	412552
				t	878984	161136	1040120	466208
A500/570	PAL	1.3	internal floppy	c	457296	65936	523232	457248
				f	412704	104184	516888	412064
				t	870000	170120	1040120	457248
A500+/570	PAL	2.0	internal floppy	c	809744	237776	1047520	795144
A500+	PAL	2.0	internal floppy	c	839688	207864	1047552	838648

* c=chip; f=fast; t=total
FD = Floppy Drive; HD = Hard Drive